NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPLEMENTING SIMULATION DESIGN OF EXPERIMENTS AND REMOTE EXECUTION ON A HIGH PERFORMANCE COMPUTING CLUSTER

by

Adam J. Peters

September 2007

Thesis Advisor: Paul Sanchez
Second Reader: Jon Alt

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE Implementing Simulation Design of Experiments and Remote Execution on a High Performance Computing Cluster		**5. FUNDING NUMBERS**
6. AUTHOR(S) Adam J. Peters		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis focused on creating an object-oriented software architecture around which tools can be created to increase the usability of stochastic simulations such as IWARS and Pythagoras on high performance computing clusters. The objective of the architecture was to enable the user to design and execute simulation experiments using a platform-independent client and server to create a common interface for various simulations. The interface input is used to select the experimental factors of interest to the research analyst and then to create the scenario files for each simulation run with minimal human intervention. To develop the architecture the current state of the art was explored, a proposed process flow was developed. This process flow was then vetted by operations researchers from several organizations. A prototype application was developed based on the software architecture. The prototype revealed great benefit in this type of tool.

14. SUBJECT TERMS Design of Experiment, Simulation, Automation, Robust Design, Database, Process Reengineering			**15. NUMBER OF PAGES** 109
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

IMPLEMENTING SIMULATION DESIGN OF EXPERIMENTS AND REMOTE EXECUTION ON A HIGH PERFORMANCE COMPUTING CLUSTER

Adam J. Peters
Captain, United States Army
B.S., University of Pittsburgh, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN
INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Adam J. Peters

Approved by: Paul Sanchez
Thesis Advisor

Jon Alt
Second Reader

Dan Boger
Chairman, Information Sciences Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis focused on creating an object-oriented software architecture around which tools can be created to increase the usability of stochastic simulations such as IWARS and Pythagoras on high performance computing clusters. The objective of the architecture was to enable the user to design and execute simulation experiments using a platform-independent client and server to create a common interface for various simulations. The interface input is used to select the experimental factors of interest to the research analyst and then to create the scenario files for each simulation run with minimal human intervention. To develop the architecture the current state of the art was explored, a proposed process flow was developed. This process flow was then vetted by operations researchers from several organizations. A prototype application was developed based on the software architecture. The prototype revealed great benefit in this type of tool.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge the following without whom this thesis would not have been completed:

My wife, Catherine, and daughter, Caitlin, for patience and unquestioning love
Prof. Sanchez for guidance and encouragement
LTC Thomas Cook for mentoring throughout my education and patience with my undergrad level questions.
My classmates especially Jeff Withee, Eddie Pena, Brian Rideout, Rusty Dash, Bob Creigh et al. for aiding in my learning process through various projects, classes and conversations.
SEED Center and the International Data Farming Workshop 14 were invaluable to understanding the design of experiment process.

And of course, many thanks to God – Gloria in Excelsis Deo.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

Simulation analysts are an invaluable asset to the Department of Defense. Each year their research potentially saves the DoD billions through more efficient operations, better procurement and decision support. The skill and experience necessary for these researchers to perform effectively can only be found in a small group of individuals. The skills necessary are normally gained through a graduate level education and the experience comes from either extended service in the military or from having long term exposure to those who have served.

The analysts provide the DoD with expert and timely answers to questions regarding equipment procurement, equipment mix and optimal utilization of the goods that they already possess. However, some of processes in the critical path to garnering these answers are unnecessarily cumbersome and error prone. Additionally, some of the knowledge necessary to expedite the process is clumped in a very small group of individuals.

Creating design point files from analyst input and allocating those files to a a high performance computing cluster are processes which when improved, will greatly increase analyst productivity. Several steps must be taken in order to improve each process. First the current state must be studied and a general model created of the process. Next the desired qualities of the end state solution will be elicited. Finally the proposed solution will be developed and prototyped.

B. OBJECTIVE

This thesis will create a generalized model of the processes required to develop a simulation design of experiment, create the design point files and the allocation those files to a computing cluster. Creating this model will aid in determining what steps of the process can be automated to the greatest advantage of the researchers. For the steps or subprocess that can be automated the model will aid in the creation of a prototype system.

The prototype will be developed using the model-view-controller design pattern. Model-view-controller design pattern separates the logic for data handling (model), the user interface (view) and the business logic (controller). This will help insure that the components of the application are decoupled so that the application can be extended with different cluster controllers, simulations and desigh of experiment algorithms.

This thesis also aims to increase the ease of using of a computing cluster to further increase the return on investment in operations researchers. A computing cluster is a group of computer processor under the control of a single interface. The interface for the computing cluster will be integrated into the prototype described above.

C. EXPECTED BENEFIT

Automating the process of creating design point files and submitting the files to a high performance computing cluster will simplify an arduous process, enabling analysts to concentrate on their areas of interest rather than working on data entry and cluster controller programming.

The target population for extracting process information and current methods of design of experiments are Operations Research students, faculty, and analysts on the Naval Postgraduate School campus, including the U.S. Army TRADOC Analysis Center – Monterey. The results of this thesis are targeted toward improving productivity for these researchers; however the overall goal is to generalize the process enough so that it can eventually be applied to any design, simulation and cluster combination.

D. ORGANIZATION OF THE THESIS

Chapter II begins with background information primarily for persons outside of the Operations Research field. It then discusses pertinent work by other authors in automated design of experiment. Chapter III describes the current process of design of experiment, design file creation and simulation cluster runs. This chapter also describes the observed shortcomings of these processes. The fourth chapter describes the system and architectural requirements necessary in order to bridge the gap between the current process and the proposed process of running a simulation at multiple design points on a remote computer cluster. The fifth chapter explains the design decisions made in order to create a working prototype of the proposed architecture. Next the process of implementing the prototype on a high performance computer cluster is described. The final chapter of the thesis summarizes the research findings and provides areas for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

The first section of this chapter provides background information mainly for readers who are not familiar with modeling and simulation. The second section of this chapter describes the current state of affairs in automating the simulation process.

A. BACKGROUND

1. Simulation

The impetus behind most scientific study is to create a better model of how the natural world works. The model is then exploited to improve understanding of the natural world in situations where the system under study cannot be tested in the real world due to time, cost or ethical constraints. In the abstract, a model is a transformation function, which turns inputs into outputs. *Factors* are transformation inputs that a researcher can change, and we are often interested in characterizing how the system output changes based on the factor values. If the model is a set of mathematical equations, we may be able to find closed-form analytical solutions to describe its input/output behaviors, but analyzing models in this way is possible only when the model is very simple. In the early years of the Computer Age we began programmatically representing models that were too complex to solve analytically. We then used the computer program to study the system of interest. A model that is a computer program is called a simulation [Law & Kelton, 2000], because it works by mimicking the behavior of the real system. As computers have become more powerful, we have been able to model larger and more complex systems using simulation.

In many real-world systems, such as where human behavior is being studied, the outcomes cannot be predicted with precision. We often use randomness to model such systems. This adds another layer of complexity – each time you work through the model, you may come up with a different set of results. Models involving randomness are called *stochastic models*, and must be studied using statistical techniques.

The use of simulation to evaluate stochastic and large deterministic models greatly increases an analyst's ability to more closely replicate the processes occurring in the natural world.

2. Design of Experiment

Proper experiments should not be run haphazardly. If the researcher wishes to arrive at valid conclusions they must set up experiments in such a way as to ensure that any conclusions derived from the results of the experiment are reflective of the model's behavior rather than the method used to study the model.

Statisticians, operations researchers and mathematicians have created volumes of work regarding how to effectively design experiments to most thoroughly and efficiently examine models. Much of the work pertinent to this thesis attempts to create a balance between two of the main qualities of the experiment design, thoroughness and efficiency. The following paragraphs present a few of the considerations in the design of experiment process . A good starting point for a more thorough understanding of design of experiments can be found in *A User's Guide to the Brave New World of Designing Simulation Experiments* [Kleijnen, et al., 2005] or *Work Smarter, Not Harder: Guidelines for Designing Simulation Experiments* [Sanchez 2006].

The most thorough probing of a model requires that each factor of interest be iterated over at as many levels as is possible. This design, referred to as a full factorial or simply a factorial, will generate a huge volume of simulations to run in all but the simplest of models. For example, consider modeling optimum automobile fuel efficiency with only three factors, speed, horsepower and fuel type. We might study speed at integer values from 30 to 75 miles per hour. Horsepower is likewise a set of integers typically ranging from about 140 to 240 in most cars. Even if we only look at horsepower in five unit increments to keep the experiment smaller, with only these two factors there are 966 (46 speeds X 21 horsepower levels) possible combinations. Each of these combinations would have to be run for each type of fuel commercially available, 87, 89 and 91 octane as well as E85, resulting in 3,864 simulation runs. If we wanted to determine fuel economy on an open road with environmental and traffic conditions, which would be a stochastic model, the design would get much more complex.

Achieving efficiency in an experimental design is important because processor time and the time required to complete research are finite and often strictly bounded for the researcher. As such, the researcher may be tempted to only set factors at a few levels, which he or she feels will create the largest effects. In doing so the researcher may end up missing a region of interest, such as a peak or plateau in the plotted relationships or bend in the curve due to an interaction. Additionally, by picking and choosing which levels to evaluate the researcher injects his or her own bias into the outcome of the experiment, possibly skewing the results.

The above covers the main considerations for deterministic models, however stochastic models require additional considerations when designing experiments to explore them. The most significant of these considerations is replication. As mentioned above, the models must be run repeatedly because they have an element of chance. Repetition enables the researcher to combine the results to determine the distributional behavior of the model. The number of replications must be large enough to give the research enough degrees of freedom to work with. Larger stochastic models can take hours, days or even weeks to run. A large number of replications at each design point may not be practical in these models.

An efficient design is the ultimate goal of the design of experiment process. An efficient design strikes a balance between the number of experiments required and the coverage of the factor space. Many methods are used, such as first looking at a course grid of design points then creating a more granular design to further investigate interesting regions of the factor space; using carefully selected fractions of a full factorial design; or using space filling designs such as Nearly Orthogonal Latin Hypercubes (NOLH) [CIOPPA, 2005], that go a great distance towards examining factors over broad ranges without running the experiment factorially. In the example above, the deterministic experiment required over thirty-eight hundred runs to cover the whole design space for three factors. Even with four additional factors added to the model, the NOLH design described by Cioppa (2002) reduces the number of design points required to thoroughly examine the design space to seventeen.

3. Cluster Computing

Once an efficient design is created, the simulation must be run once with each set of data in the case of a deterministic model. More frequently, however, we are dealing with stochastic models so the simulation must be run repeatedly at each design point in order to derive the most likely behavior the model. The processing time for just one run of a moderately complex model is often measured in hours. The smallest NOLH experimental design requires seventeen runs. If each run requires only an hour, but you require thirty replications to create a statistically valid data set then the experiment will take over twenty-one days on a single computer.

An experiment's processing time can be cut substantially by using cluster computing. A computing cluster is a group of computer processors controlled through a single interface. Computing clusters increase computing power and computation speed through one of two methods. First, they can process one job faster by splitting up the work between the available processors. This method, referred to as parallel computing, is difficult to achieve based on the high level of inter-processor coordination often found within a single run. The simulation developers would have to intentionally write the application code to take advantage of parallel processing, and many problems cannot be effectively decomposed to take advantage of parallelization. The second method for increasing computation speed is to process many separate jobs at the same time by porting the jobs amongst the available processors. The simulations of interest to this thesis are not designed to run using parallel computing, so we will focus on the latter method.

A modest cluster of ten processors, using the method above, would cut the processing time required to a tenth of what it was. The twenty-one days spent waiting for results would is cut to just over two days. Even though this is just a linear decrease in processing time, it can be very significant. Researchers will find it more practical to create and run an efficient design when they can have the results in a matter of days rather than weeks or months. Add this in with Moore's Law and the effect on processing simulations is greatly compounded. However, based on Gottbrath, et al.'s [1999] work we can conclude that if the simulation is large enough, it may be best to move with deliberate speed in putting together your model and design of experiment.

B. RELATED WORK

1. Software Packages for Design of Experiments

Software to automate the design of experiment process was developed and described almost forty years ago [Kennard, 1969]. Efforts of this sort have been ongoing since. The results can be lumped into three general categories. Software packages designed specifically for this purpose, such as the one referenced above, or add-ons to software to aid in design of experiments, comprise the first category. The second group is simulations with some sort of built in design of experiment capability. Simulation packages that can accept input from a separate data source round out the categories. While each type has many positive attributes, they lack the ability to provide efficient design of experiment support to the Department of Defense simulation community.

Software packages designed specifically for design of experiments as well as statistical software such as JMP [JMP], Microsoft Excel plug-ins such as Crystal Ball [Crystal Ball], or applications built into spreadsheets [Sanchez 2005] make up the largest group of electronic aids in design of experiment. They typically require the user to input the high and low limits of each factor plus the number of digits of precision to use. The application either has a predetermined design, such as Kennard's (1969), or has a set of designs from which the user must choose. The application then algorithmically changes the factor values, based on the design. The design output is then displayed in a tabular format for the user to apply to his or her simulation model.

These applications represent an innovative leap ahead from manual methods, but lack an interface with the model creation process. As a result the creator of the model and corresponding design of experiment must painstakingly work through each model input file and insert the factor settings by hand. This leads to one of our primary motivations for this work. The experimental designs are algorithmic, so can easily be codified. Most of the designs are openly published with no claim to use-rights, so they can be used with any model to create a set of input files for simulation runs.

Simulations with design of experiments built in provide a nice tool for the simulation in question, but are of questionable value at this point in time. In a recent review of available simulation software [INFORMS, 2005] only seven out of the fifty-eight products, from forty-eight vendors, claimed some sort of design of experiment capability. Upon review of the product documentation (e.g. TreeAge [TreeAge], SIMPROCESS [SIMPROCESS]) the experimental design utility is actually a batch parameter input module. The products allow you to enter your design of experiment, but they do not actually aid you in creating the experiment design.

Products such as Arena [Arena] and Process Modeler [Process Modeler] have come to prominence in the business world for optimizing workflow through process modeling and reengineering. The requirement for extracting experimental data from a spreadsheet or database was identified and has been included in these products for the past several years. As a result efficient designs of experiment can be created in products such as Sanchez’s [2005] and ported to these simulations. There are two major downsides to this method. First, you are locked into proprietary simulation software. Many different types of simulations are necessary in the military to answer the types of questions important to us. This leads to an issue of time wasted by a researcher learning how each simulation’s process of extracting data works. In practice, there is a fair amount of programming necessary. The larger problem, in the case of the Department of Defense, is scalability. The models of interest to the DoD are quite large in scale and/or high in resolution. The time it takes to run a simulation is often measured in hours and sometimes in days or weeks. The majority of simulation packages in question have no utility to launch jobs on a computer cluster so each simulation is run serially on a single computer processor.

A caveat to the information garnered from the INFORMS [2005] survey of software is that it does not include simulations used to study the combat models of interest to this research. However, based on observation of the software that is used, the military simulation market is in the same state as the business simulation market. There are many companies, which provide many single function products with little to no design of experiment or cluster computing capability. This thesis was specifically tasked by the US Army TRADOC command at Monterey (TRAC/MRY) to fill this gap for their analysts.

2. The Tiller

The Tiller, developed by Referentia Systems Incorporated [Referentia, 2007], is notable in that is does combine design of experiment, cluster computing and batch processing in a way that was not previously conceived.

The Tiller was developed for Project Albert, a Marine Corps Warfighter Lab “research and development effort whose goal is to develop the process and capabilities of Data Farming [Project Albert, 2007].” The Tiller was primarily designed for the Map Aware Non-Uniform Automata (MANA) and Pythagoras simulation packages. The Tiller provided a graphical user interface for factor selection, selection of factor ranges, design point file creation for batch processing, and the ability to send simulation jobs to a remote computing cluster for processing.

The Tiller is a step in the right direction. However, it was completed within a tightly defined set of constraints, resulting in a highly specialized, inextensible application. Some of these constraints include limiting available factors through a ‘roadmap’, and the target simulations.

The ‘roadmap’ is a hard-coded list that provides the Tiller with the simulation’s farmable factors, i.e., those that are pre-judged as suitable to vary. In turn, the Tiller lets the user select only these factors. The ‘roadmap’ was a tradeoff made between thoroughness and ease of use. An exhaustive list of parameter values in a model can range from a couple hundred to tens of thousands of unique inputs. In an effort to mitigate the size of the listing, the software design errs on the side of brevity. The difficulty here is that there can be no consensus on what might be a factor in an experiment. A researcher examining the effects of unmanned vehicles on squad tactics may consider 'soldier enemy detection range' as an immutable constant, while the researcher examining the effects of chemical protective masks might vary the detection range as one proxy of the effects of wearing the mask.

The target applications for Tiller use Extensible Markup Language (XML) to describe their model inputs for simulation. Several other simulations of interest to Department of Defense researchers, including the Infantry Warrior Simulation (IWARS) and Combat XXI use XML. If the Tiller had been created in a more general fashion, for example with no requirement for roadmaps, it could have been used to design experiments for additional simulation packages. However, although the Tiller is a great innovation for its target simulations and experimental designs, its rigidity limits its utility in the broader field of simulation analysis.

3. Conclusion

Overall, the simulation software market is trudging forward by adding some labor saving design of experiment data entry and batch processing utilities to their products. This effort is hampered by two main efforts that simulation vendors appear to be using to try to gain competitive advantage. These efforts are modeling and simulation visualization tools, and creating tightly coupled 'answer to everything' software packages.

The main thrust of innovation in simulation software today is in adding in two- and three-dimension visualizations of simulation models. One result of this is simply flash. The other can be quite useful when used correctly. The process of building models and running simulations can seem quite abstract to those with no background in the field. When 'The Boss' gets the results of a multi-month process simulation effort back and all she sees is a spreadsheet and an analyst's advice it can seem less than worthwhile – especially since she often feels that she knew how it was going to turn out from the start. When the boss is presented with a flashy three-dimensional visualization of the current process, the optimum process, *plus* the spreadsheet to show her the numeric difference, the effort seems much more worthwhile. While this may help “sell” the results, creating the visualization is non-productive time for the analyst.

Visualization of models can make model building easier through ‘drag and drop’ placement of entities, entity reuse, decreased abstraction, and visual verification of the model. These tools can open the process of modeling and simulation up to a broader audience because persons less adept at programming can now build quite complex models. This is both a boon and a curse. While visualization makes the process easier, it does not necessarily make it better. If the person researching the problem hopes to come up with useful answers from the research, the model must be developed methodically. A person untrained in the science of programming, simulation or statistics may find the numerous options, such as choosing between a Poisson distribution and normal distribution for some process, to be of little use or confusing. As a result the modeler will select default options, or simply guess at what is best. This leads to invalid models (though the researcher would not know) and the results of these models can actually cause great harm to the organization that they were developed to help. This should not be construed as an elitist viewpoint held by the modeling and simulation community to preserve egos or jobs. The use of modeling and simulation has repeatedly proven its worth. The prestige of those trained in these techniques can only increase with increased use. However, any tool incorrectly applied is useless and often counterproductive.

The golden goose of commercial software development is to create a product that thoroughly and completely meets all of the customer’s needs. With simulation software the goal is no different. The problem with this is that new applications for simulation, and other software, are developed every day and that there is no one true set of standards for all models or simulations. Innovation is good, but one company cannot hope to provide all the answers. Software companies still strive to do so, though. A company may develop software that is especially good for simulating chemical dispersion in disastrous spills. When customers request additional functionality from the software, say running the scenarios on a high-performance computing cluster, the software company typically folds this function into the current software package. Unfortunately, cluster computing controllers come in many varieties so it is unlikely that the simulation software company will create a product that will work with all cluster controllers – the software that dispatches and coordinates threads or processes on individual processors in the cluster. The product will likely work well with a few cluster controllers, until a few months later when the controller software is upgraded or patched.

A better answer is to provide an open interface to the simulation software that can easily be accessed to run programs using the client's cluster controller. Adding automated components to work with a few of the most common controllers is worthwhile as long as the user can still access the simulation processing software through other cluster controllers.

III. ANALYSIS OF CURRENT TECHNIQUES

This chapter describes the current process of designing and executing simulation experiments. The information necessary to develop this chapter was described in informal interviews and conversations conducted from November 2006 through March 2007 at the Naval Postgraduate School; from the Simulation Analysis (OA4333) class lectures and readings; and from conversations and work conducted during the 14th International Data Farming Workshop. Based upon the collected information, the process was modeled using a Unified Modeling Language activity diagram (Figures 1 and 2).

The model is not inclusive. Much of the thought behind creating a well designed experiment is codified [Kleijnen, et al., 2005]. However, some of the considerations have a great deal of variability based on the researcher’s goals for the experiment [Kleijnen, et al., 2005]. Finally, some portions of good experiment design are tacit, so are best learned through doing [Rusco, 2003], and therefore difficult to codify. The model is composed of those portions that are explicit and includes as much of the variability as possible.

A. DESIGN OF EXPERIMENT

Design of Experiments (DOE) is a thoughtful process and cannot be taken lightly. If Operations Research is a science, then the methods used in obtaining answers must be no less rigorous than those used by pharmaceutical researchers and chemists. The result of a poorly designed experiment to determine a proper unmanned vehicle mix is potentially as disastrous as an improperly designed experiment for a drug clinical trial.

1. Current Method

The model (Figure 1) begins with several assumptions based on the scope of this research. We assume that the researcher has: 1) framed the research question; 2) selected the simulation software based on the research question; 3) selected appropriate measures of performance or measures of effectiveness; 4) determined the need for response surface complexity; and 5) built a system model that simulation software can process.

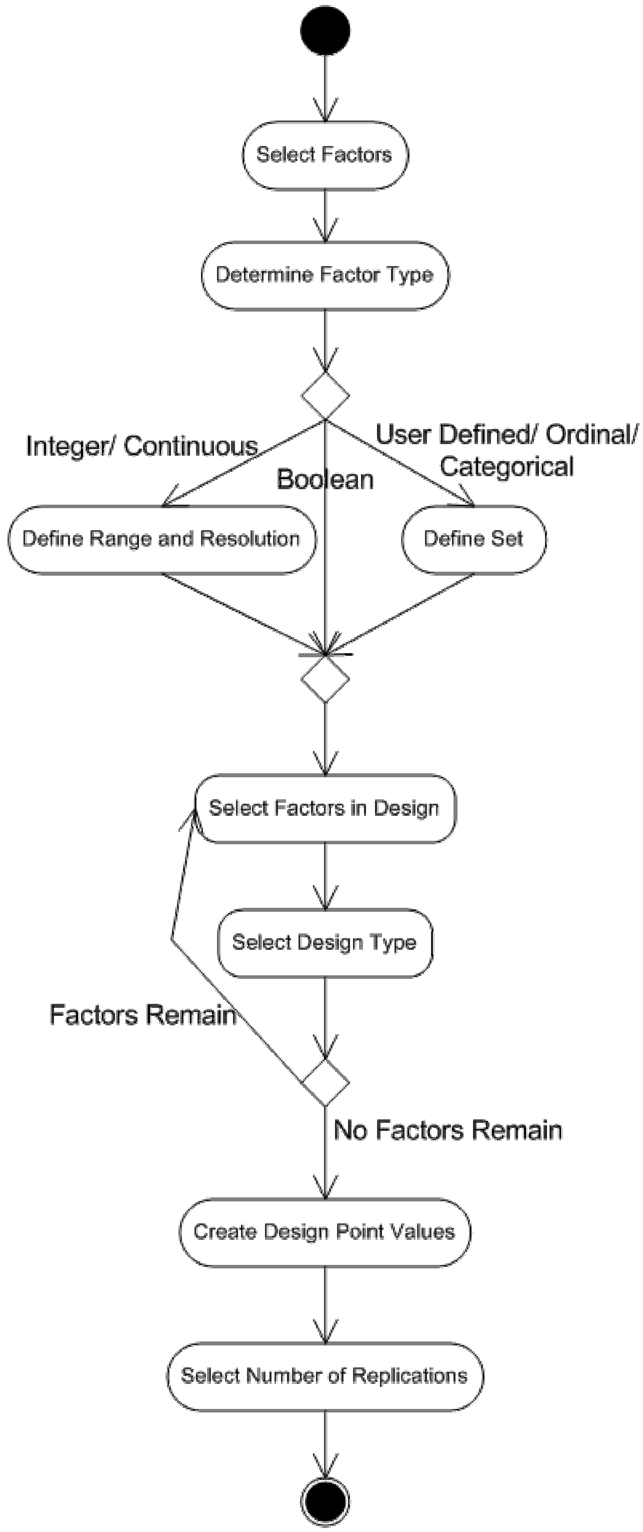

Figure 1. Current DOE Process

At the start of the Current DOE Process (Figure 1), we have a working model that is validated to run on the simulation of choice. The researcher created the model with parameter values that he or she believed to be valid based on experience, research or a combination of both. Model validation is another area of current interest in the simulation community [Law & Kelton, 2000], but outside the scope of this thesis.

a. Select Factors

In the first step of the process, the researcher identifies and selects factors of interest for the experiment. Factors are qualitative or quantitative inputs of a model, model element or group of model elements that can be varied and are hypothesized to have some effect on the system's behavior. Factors are also referred to as variables.

Conceptually identifying the factors of interest requires extensive familiarity with the simulation and model. Identifying the factors conceptually requires that the researcher understands the system well enough so that he or she can identify what inputs might have an effect on the output. Typically, the person that creates the model of the system will be the same person that designs the experiment to study the model. This is not always the case, though. Some systems under study are large enough that collaboration between several modelers is essential to create the model accurately, completely and in a timely manner.

The selection of factors can be completed in several ways. Sometimes the research sponsor thrusts the factors of interest upon the researcher, such as in a study of the effect of different equipment packages on unit performance where the attributes of the equipment are already set. The factors may also be selected by the researcher, based on his or her experience, as the attributes that have the greatest impact on the measures of interest. For example, speed is generally accepted to have an impact on fuel economy, so most analysts would include speed as a potential factor if fuel economy is the measure of interest. The selection of factors may also be a collaborative effort between any of the following: researcher(s); modeler(s); subject matter expert(s); and the research sponsor(s).

Limitations in simulation software can complicate the factor selection process. In some cases there is not a directly attributable input for a factor of interest. That is, a simulation is built to allow for certain elements to interact. Each element has some number of attributes. If the researcher wishes to study something other than these attributes, he or she must either aggregate the available attributes to create the effect or use a different attribute as a substitute to that effect. To illustrate, consider an automobile manufacturer who wishes to study the effect of adding a spoiler to several prospective car designs on fuel consumption and handling. The simulation software that the manufacturer uses does not include a utility to add a spoiler. However, the simulation does have 'vehicle drag' and 'tire traction' parameters. The change in drag and added traction from the spoiler, as calculated by the engineers, can thus be combined with the substitute parameters 'vehicle drag' and 'tire traction' to get the same effect as adding a spoiler. This allows the manufacturer to validly answer the research question they have posed without purchasing or creating new software specific to the problem.

At the end of the factor selection process a list of factor names is recorded by pen and paper or in a spreadsheet of individual or commercial design. We will assume the use of a spreadsheet like Sanchez's [2005] to illustrate the rest of the process.

b. Determine Factor Type

The second step modeled in Figure 1 is to determine which type of factor you are dealing with. Generally, factors can be classed as *continuous, ordinal,* and *categorical*. Continuous data are numeric and correspond to real numbers. Ordinal data may or may not be numeric but have an ordering property, e.g., classifying a student as Freshman, Sophomore, Junior, or Senior. If they are numeric, they correspond to integers. Even if they are not numeric, they can often be mapped to integers in a meaningful way. Categorical data (a.k.a. nominal data) have discrete categories but no ordering property, e.g., Color = {Red, Green, Blue}, or Gender = {male, female}. Defining the type of factor is important as it begins to define what types of experimental designs will work best for these factors. For example, continuous and ordinal factors can be studied with NOLH or factorial designs, while a categorical factor can only be studied factorially and is often crossed or blocked against the design used for other factors. Note that although Boolean data are just a special case of categorical data, we treat it separately in Figure 1 because the binary nature of Boolean data places significant constraints on the types of designs that can be applied.

The work that the researcher does in this step is based on his or her experience with the model and the system under study. The DOE spreadsheet that contains the names of the factors from the previous step can now be annotated with the factor types.

c. Define Range and Resolution

For continuous and integer factor types the next step is to define the upper and lower ranges and the size of the steps between each factor setting or the precision of the factor settings. The researcher defines the ranges based on the experience he or she has with the system under study or from input from a subject matter expert on what is possible and practical in the system. The ranges may be dictated by the research question if the researcher is working for another party. Finally, the range may be determined based on the results of a previous set of simulation runs. A previous experiment may have used a coarse screen to find the most interesting response areas of the model. Once the researcher has narrowed down what ranges yield the responses of greatest interest he or she may wish to create a tighter grid around this area to explore it more thoroughly.

The resolution of a factor refers to either the decimal precision that the researcher wishes or the size of the step between each factor setting. The decimal precision is only defined for continuous factors. Knowledge from the researcher's experience or subject matter expert input defines a suitable number of decimal places. The step size can be defined for either continuous or integer factors. Step size could either be used for factors that have a very large range or for factors that have some valid numerical constraint such as in a study of

Multiple Launch Rocket System ammunition optimization in some war scenario where the rockets can only be issued in groups of six because they come in a preloaded pod, but can be fired one at a time.

For integer and continuous factor types, the DOE spreadsheet begins to take shape. The ranges and resolutions are entered below the factor names.

d. Define Set

The remaining factor types can be handled as categorical for the purpose of his research. The researcher, again based on his or her experience or subject matter expert input, must define the available settings for the factor of interest. Booleans can be defined in many ways (e.g., True/False, 0/1). The researcher must know how their particular simulation defines Boolean, and then document it. In any case, for DOE purposes a Boolean is just a categorical factor with only two possible categories. Ordinal and nominal factors can often be treated the same in DOE. In the final design, the categorical factors will either be randomly distributed with the numeric factors if there are a large number of categories they will each be run against a full complement of the numeric portion of the design (crossed or combined). These considerations bring the researcher to the next step of the process:

Factors other than integer and continuous are put to the side of the DOE spreadsheet with their defined sets.

e. Select Factors in Design; Select Design Type

At this point, the researcher is done figuring out *what* is going to be in the experiment and begins determining *how* to fully explore the effects of the factor on the model. This also seems to be a point at which tacit knowledge is necessary in order to create a truly efficient experiment. Kleijnen, et al. (2005), discuss at length the process of selecting the appropriate design based on number of factors and response surface complexity. The process is depicted as iterating between selecting factors in the design and selecting the design type for cases where there is a primary design grid, say NOLH, that contains most of the factors, then crossing it with a gridded design of the remaining factors to yield a combined design. The process continues until all factors are incorporated in the design.

f. Create Design Point Values

This step of the process can occur in concert with the previous steps and either before or after the following step, Select Number of Replications. The order has no real bearing on the number of replications but it might be useful to have a grid of the factor values while creating combined designs. This step is generally completed by some sort of automated system, which takes the results from the previous steps, particularly the factor range and resolution, and applies it to some algorithm to generate a set of factor values for each simulation run required by the design. The automated system generally outputs the data in a spreadsheet format with each of the factors on one axis and the run number and enumerated factor settings on the other axis. The full set of factor values for one simulation run is referred to as a design point.

The DOE spreadsheet scales a design, initially in standardized units, for each of the factors by using the ranges and resolution in order to come up with the design point values. Then, based on the researcher’s design decisions for the remaining factors, the design points may need to be replicated for each category of the remaining categorical factors.

g. Select Number of Replications and/or Termination

The final step in creating a DOE is to determine the number of replications and, for steady-state simulations, how long the simulation will run. As mentioned before this step can occur at an earlier stage of the process, however the number of design points created in the previous step may have some impact on the decision made during this step. In a stochastic simulation, it is necessary to run a model multiple times in order to determine the amount of variability in the system. The more times you run the simulation, the more degrees of freedom you have, resulting in improved statistical estimates. Practically though, the simulation replications are limited by the simulation processing time. The maximum number of simulation replications is the time available divided by the product of the time it takes to process one simulation and the number of design points. This is usually complicated by the fact that run times can vary significantly for different design points.

For steady state simulations, a terminating point must be selected. These simulations may suffer from initial bias, the affect of the initial conditions of the simulation on the results during a warm-up period. As a result, the simulation must run long enough that the researcher can remove some appropriately large set of initial results and still have a statistically valid set of result data.

These decisions are not used by the DOE spreadsheet but are annotated by the researcher in order to document his or her overall design decisions.

2. Analysis

Overall, the current process is quite efficient. Much of creating the DOE is a mental process rather than a physical process so it is difficult to make it more efficient other than by formalizing the process, such as in using Study Question Methodology [Rauhat, 1999].

Some efforts have been made to make the design point creation step more efficient. Researchers have formalized and codified some designs, such as factorials, fractional factorials, and Nearly Orthogonal Latin Hypercubes, so that others do not have to create a design and calculate a 'goodness' measure each time they wish to conduct an experiment.

The two areas that could use improvement in the process as described are documentation and integration with later processes. There is currently no formal way for researchers to collaborate asynchronously on a DOE other than physical transfer of the design, such as by e-mail or an ftp site. A system where researchers could make and document their decisions would benefit peer researchers and pedagogy.

Integrating DOE and the creation of design point files will have a slightly negative effect on the DOE, but will vastly improve the process of creating design point files. DOE will take more time, as the researcher will work with the physical, rather than conceptual, model, so his decisions can be applied immediately rather than in a separate process. Integrating these processes and automating some of the steps in design point file creation will result in a net gain in productivity for the researcher.

B. DESIGN POINT FILE CREATION

1. Current Method

Much work must be done to make the simulation software able to process the researcher's well-designed experiment, but most of that work is just tedious data entry. The first step is to create individual model files corresponding to each design point. Figure 2 shows this process.

At the start of Figure 2, we assume that there is a validated working model and that the researcher has created a DOE based on that model. The DOE is encoded in a spreadsheet and the model file is available either as a local or remote resource on a computer. The model file contains nominal values for each factor based on the modeler's assumptions about the current state of the system. We will refer to such a file as the 'base case file' through the remainder of the thesis. Each design point in the DOE is applied to the base case file to create a corresponding design point file where the factor values specified for the design point are substituted for the base case factor values.

a. Locate Factor

Once the researcher has opened the base case file, he must find the factor of interest within the file. Identifying the factors of interest physically within the model schema requires extensive familiarity with the simulation and model. Often the person who creates the model of the system will be the same person who designs the experiment to test the model. Some systems under study are large enough that collaboration between several modelers is essential to create the model accurately, completely and in a timely manner.

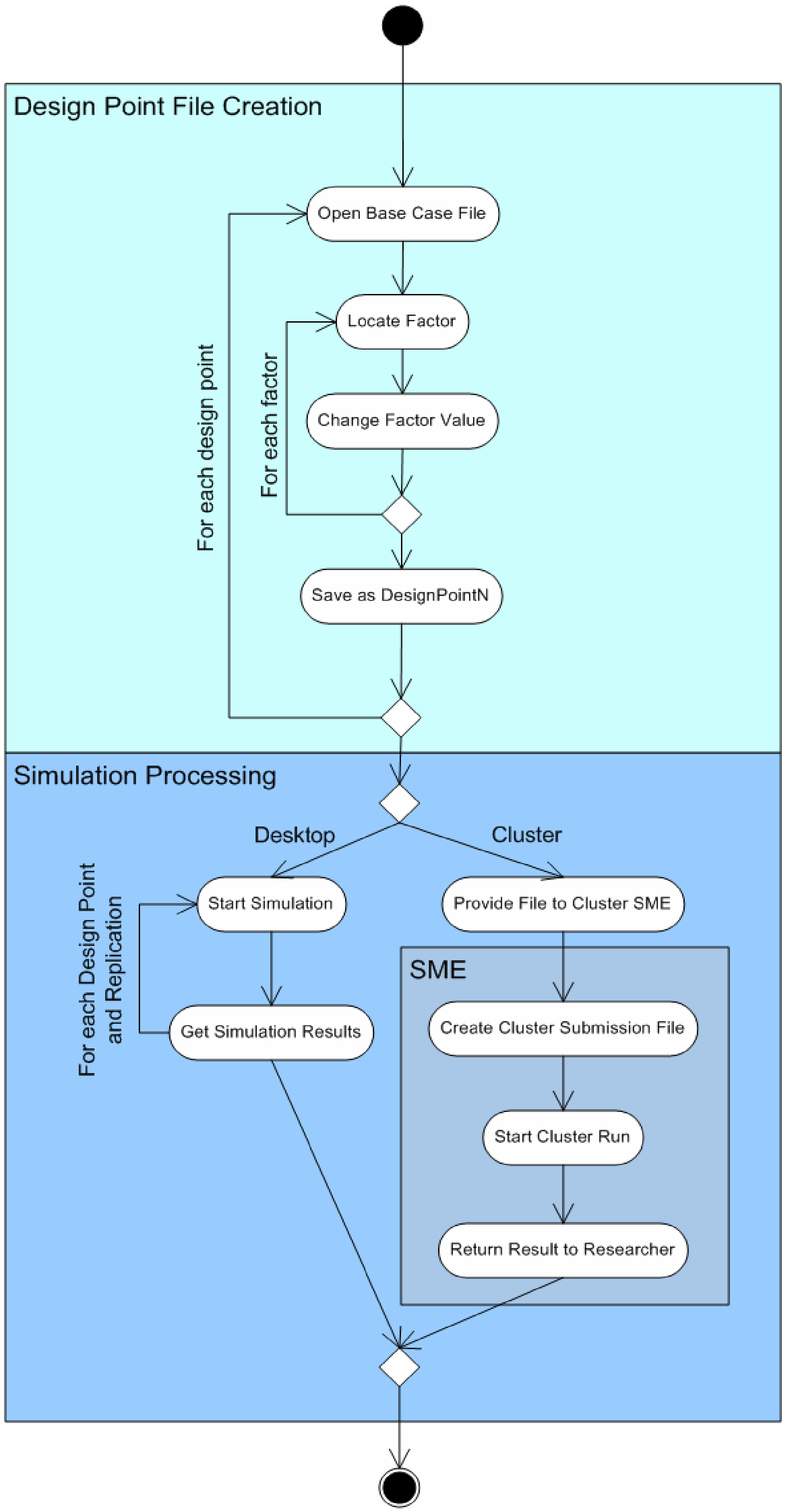

Figure 2. Current Design Point File Creation and Simulation Processing Process

The size of the model file and inconsistent naming conventions are the main obstructions to physically locating the factors of interest. Three primary factors affect model size, as measured in bytes of data rather than absolute data file size: the resolution of the model; the requirements of the simulation; and the number of objects modeled in the system. Model files increase in size as each of these factors increase, making it difficult to find the factor of interest. For example, a moderately sized Map Aware Non-Uniform Automata (MANA) simulation model with four agent types represented in ten 'squad' objects with varying attributes has approximately ten thousand lines of input. Each of the squads is represented by between four and eight hundred bytes of data. In MANA models, many of the data pieces are labels. Even if half of the data are labels this still leaves the researcher to comb through up to four hundred lines in order to find the particular factor of interest.

MANA's models are encoded using Extensible Markup Language (XML), but simulations that require database input are at least as complex. The Assignment Scheduling Capability for Unmanned Aerial Vehicles (ASC-U) simulation relies on data in fifty-three tables in a relational database. Neither MANA nor ASC-U are unusual in this regard.

The naming conventions used in the simulation models are at least as prohibitive as the size of the files when it comes to rapidly locating factors of interest. Some elements are named quite clearly, but others are nonsensical, such as 'ResOrgUnknown' in MANA and 'DFHitobject' in ASC-U. The software programmers may consider these names to be self-descriptive, and omit documentation. These elements may be very important to an experiment, but to anyone coming behind the programmers the names are gibberish. MANA and ASC-U are not unique in this regard. The problem of non-obvious nomenclature occurs in many software applications.

b. Change Factor Value

Once the factor of interest has been located within the base case file the researcher refers to the DOE spreadsheet to determine the factor value.

This value is entered into the model file and the researcher may annotate the spreadsheet so he or she knows that the work for that step is complete.

The process is repeated until all factors in the design are exhausted. The researcher determines the next factor of interest, locates that factor in the base case file, as above, and then changes the value to the design point value. This process may seem linear, however each time the researcher needs to locate a factor of interest he has to begin again at the top of the (ten-thousand-line) file and try to reconcile the conceptual file name in his or her mind with the physical factor name in the model.

c. Save as 'DesignPointN'

A base case file's factors of interest are now altered to reflect one design point in the DOE. The researcher saves the file with a unique name. Typically, the file name reflects the arbitrarily numbered design point from the DOE (e.g. simulationName_modelName_1, simulationName_modelName_2, …, simulationName_modelName_N).

The overall process of design point file creation repeats until each set of design point values have been merged into the base case file and saved with a unique name.

This step and the previous step, Change Factor Value, are fairly simple, but tedious. Both steps provide a prime area of entry for transcription errors into the experiment. If performed manually, incorrectly factor value entries will create spurious results, leading the researcher to misguided conclusions about the system under study. When the file is saved it is all too easy either to save over the original or to misnumber the file. This results in the incorrect pairing of input factor values and output measures of interest. If the effect is noted, recovery is just as difficult as creating the files in the first place – the researcher has to go through each file to find which were correctly edited and saved and which were not.

2. Analysis

As mentioned above there are only a few steps to complete this process. Each step is fairly straightforward. The factors are located in the same location in each file, but it is easier to open one file at a time than it is to open, save and keep track of tens or hundreds of such files. The values for each design point have already been determined so all the researcher has to do is transcribe them from one place to another.

Computers outperform humans on many tasks. Tasks that require low variability, high accuracy, high speed and little creativity are ideal candidates for computing solutions. The design point file creation process has all of these attributes. Humans perform very poorly in such circumstances. A task with little variety or creativity drives a human operator toward lower accuracy, and humans cannot come close to the speed at which a computer processes data.

C. SIMULATION PROCESSING

1. Current Method

The final process of interest to this thesis is the actual processing of the design point files with the simulation software. At the start of this process we assume that there is now a set of design point files that are valid in format and represent the marriage of the base case files and the DOE. The other assumption is that the researcher has the simulation software available on a desktop computer, a group of desktops (e.g., a computer lab) or has access, either locally or remotely, to a high-performance computing cluster with the simulation software resident.

As modeled above (Figure 2), this process generally takes one of two tracks. The researcher can run the simulation on a desktop computer or with a high-performance computer cluster. First, the desktop path will be explored, then the high-performance computing cluster.

a. Desktop Processing

This method of processing also has two main variants. In the first, the researcher runs the simulation on his or her office desktop computer or a spare computer that is not often used. In the other method, the researcher runs simulation on a bank of computers such as in a computer lab. The second method is often completed late at night or on weekends so as to not draw the ire of the researcher's fellow computer lab users. Both methods follow the same process and net the same results, however using multiple computers may hasten the completion of the process.

(1). Start Simulation. The researcher opens the simulation software then runs the simulation with the design point file. Some researchers are lucky enough to use simulations that have the utility to run replications from a single design point file. They either output the results in sequentially number files or concatenate the results onto the end of a single file. Alternatively, if the researcher knows how to write batch files, which is not common, he or she can create one to run multiple replications.

(2) Retrieve Simulation Results. The simulation results are usually returned in one of several formats: a text file of comma separated values; a formatted text file; or a database. The output filename is either specified when the simulation is started or created by the simulation.

These two process repeat as many times as necessary to exhaust all of the design point files and replications required by the DOE. Like creating design point files, the process is fairly straightforward. Also like design point file creation, the process is tedious. It is error prone in the same vein and detracts from the researcher's ability to do useful work.

Let us explore the time it takes to complete a simple scenario where the DOE has five factors and each design point must be replicated ten times. If the researcher chooses a nearly orthogonal Latin hypercube there are seventeen resultant design points. If the simulation averages a mere five minutes to run to completion, then the total time for processing is 14.17 hours (17 design points * 10 replications * 5 minutes / 60 minutes per hour). This is not a vast amount of time until you consider the equation's assumption that the researcher is always present at his desk ready to start the next simulation as soon as the previous one ends; that he keeps perfect track of how many times he has run each design point; and that he keeps perfect track of which design points he has run.

The time it takes to complete a simulation set is cut linearly by using a bank of desktops. If the researcher uses five desktop computers, then, with all prior assumptions, the simulations will be complete in one-fifth of the time or 2.83 hours. This is much better, but adds in the assumption that the researcher can move back and forth between the computers with no delays. It also complicates the task of keeping track of which design point files have been run and how many times.

b. Cluster Processing

High-performance computing clusters are available for use in the Department of Defense. Most of the knowledge required to utilize these tools is specialized and possessed by only a limited number of persons. While the researcher may have the knowledge required to make runs on a high-performance computing cluster, it is uncommon. As such, the process described here will be from the perspective of a researcher who does not possess this knowledge. This process requires an additional resource, a person who knows how to run the high-performance computing cluster, which we will call the CSME (Cluster Subject Matter Expert).

(1) Provide Files to CSME. This step presupposes that the CSME is willing to take the time and is available to process the researcher's experiment.

The researcher makes the design point files available to the CSME either by e-mail, portable storage device or shared network resource. E-mail may be impractical as the files can be large and there may be tens or hundreds of them. Likewise, a shared network resource may not be available.

(2) Create Cluster Simulation File(s). The researcher is now out of the loop and has little control over when the simulation is run unless there is a subordinate-superior relationship involved. The researcher will regain control of the experiment when the results are returned to him or her.

The CSME's job is fairly simple at this point. Each computer cluster controller has a particular way of presenting it with a job for processing; we will call it a submission file. The submission file varies slightly from controller to controller, but is often a very simple text file with only a few lines. In general the submission file contains the command line instruction(s) necessary to start the simulation, and any variable information to add to the instruction set. Variable information may include items such as the design point file name, how many times to repeat the command line instruction to start the simulation, where the simulation output should go, where to put error information, and where to log information regarding the processing of the files.

(3) Start Cluster Run. In this step, the submission file created previously is run either through a graphical user interface or by a command line entry. Prior to starting the simulation the CSME needs only to ensure that the design point files are available to the cluster.

If the submission file and design point files are appropriately programmed, this step generates the raw data representing the results of all of the researcher's simulation runs.

(4) Return Results to Researcher. This step begins when the simulation software has completed processing the design point files and outputs the data into the designated files. If the design point files and submission files were programmed correctly, then the results will be listed either sequentially in one file or in a set of sequentially numbered files. In either case, the researcher needs to be able to determine which results are from which design point.

This step is also fairly simple. The CSME collects the result file or files from the disk location where the cluster was told to output them. The files are then transferred back to the researcher through one of the same methods used to send the files to the CSME. The same problems of file size and resource availability that hamper the Provide File to CSME step hamper this step. The CSME motivation may also delay this step. The CSME is often a busy researcher also. As a result, the CSME's priority may not be to repeatedly check the status of some other researcher's simulation, nor to collect the files and transmit them to the other researcher.

2. Analysis

a. Desktop Processing

It should be obvious from the discussion of the process that running simulations on a desktop computer or even a set of computers is tedious at best and enslaving at worst. The researcher is, for all intents, tied to the computer for the duration of the process. While the simulation is running the researcher probably does not wish to use the computer for other functions. Doing so would conflict with the simulation experiment – any time spent on other process takes away from the computer's ability to rapidly produce simulation results.

Small, quick simulations with either batch scripts or some innate replication ability and one-time simulations for model verification are still reasonable on a desktop. Larger jobs should be completed in some sort of automated fashion on a remote resource.

b. Cluster Processing

High-performance computing clusters are available for use in the Department of Defense. Most of the knowledge required to utilize these tools is specialized and clumped in a limited number of persons. As a result, many simulations are run either serially on one computer or in parallel on a set of handy computers such as a computer lab late at night. This method also involves a great deal of tedium and can lead to errors. Typically, the researcher uses a spreadsheet to keep track of which computer is processing which design point as the results do not indicate the inputs other than possibly by file name.

The combination of learning how to program cluster submission files and the commands to invoke the simulations from the command line can be cumbersome to researcher, and in reality is usually of little interest to the researcher or analyst. As a result the researcher may just 'go with what he knows' rather than using new simulations or attempting to utilize a high-performance computing cluster.

The few people that are most interested in the use of cluster computing become the subject matter experts in its use. When other researchers wish to process large jobs they need seek out the experts or send them the files for processing. This introduces two extra steps in the critical path between asking the research question and finding an answer. First, the expert user must find the time to prepare the file for submission. Then he or she must get around to sending the simulation results back to the interested party.

Rote steps that are unambiguous, though hard for the uninitiated to remember, consume much of this process. There is a clear need for an automated interface to ease the process of running a simulation and collecting the results.

D. CONCLUSION

Converting the researcher's DOE decisions into design points has been made fairly trivial. Injecting those design points into the model for a simulation run is not. Currently the work is done manually or by writing one-off programs that comb through tens to hundreds of thousands of lines in text files to find the dozen or so lines with the factors of interest. Once the lines are found the values are changed to match a design point then the file is saved under a new name and set aside for processing. The effort to update values in database driven simulation is no less obtuse. Again, the researcher must comb through dozens of often cryptically named tables to find the correct row and attribute value to update.

This process is neither seamless nor of value to the researcher, other than as a necessary step towards solving the research problem. The experimental design is created in one application then transposed line by line into a design point file for processing using a third application, the simulation. The multiple recording and rerecording of data is error prone if done manually. When the size of the task and the enormity of the questions at hand are taken into account, mistakes are very likely and could have catastrophic results. The researcher is spending far too much time completing data entry tasks rather than carefully constructing models or analyzing simulation results.

The next chapter will develop a set of requirements for a proposed system, which remedies the shortcomings of the current system as well as the issues brought up in Chapter II. The activity diagrams that model the processes described in this chapter will be modified and merged to support the requirements development and system engineering process for an integrated solution. Furthermore, a data model will be developed using the data elements exposed in the models and discussion from this chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SYSTEM REQUIREMENTS

The previous chapters describe the current state-of-the-art in DOE and simulation processing automation, then map how the process of DOE, design point file creation and simulation processing are typically completed. This chapter defines the characteristics of a system to automate these processes from the information gathered in the earlier chapters. First, the user requirements of the automated system will be evaluated. Next, we will look at the system requirements from the developer point of view. A new process model is then created based on these requirements, the models from the previous chapter, and stakeholder feedback. Finally, the data model for the system is developed.

A. USER ARCHITECTURE REQUIREMENTS

From the user's point of view, there are three quality attributes of primary importance to the system. They are Usability, Accessibility and Security.

1. Usability

According to The Free On-line Dictionary of Computing [2007] usability is:

> The effectiveness, efficiency, and satisfaction with which users can achieve tasks in a particular environment of a product. High usability means a system is: easy to learn and remember; efficient, visually pleasing and fun to use; and quick to recover from errors.

While this thesis is not concerned with making a 'fun to use' system, it is concerned with making a difficult process easier. If it accomplishes that, but creates a system that is difficult to use, the Operations Research community is no better off. Three aspects of usability that this research sees as essential are training, error recover and system feedback.

a. Training

The user should be able to operate the system with little to no system specific training. The system will not require any significant training to operate. The system will be self explanatory, offer on-screen or linked assistance to the user and use standard terminology throughout. The user will not have to search for functions or information.

The user will not have to learn how a high-performance computing cluster operates, just that a high-performance computing cluster can process many of the user's simulations at one time. High-performance computing clusters go underutilized because researchers do not understand how to use them.

This system requirement assumes that the user is trained in DOE as it pertains to simulations. It also assumes that the user understands the standard terminology and designs for DOE.

b. Error Recovery

The user is warned when invalid data is entered. The system checks data at the point of entry rather than waiting for errors to occur at run time. Data entered by the user will be validated for data type (e.g., numeric, date), size (e.g., password length), uniqueness (e.g., simulation name and version) and presence (i.e., required fields). When a validation error is detected, the user will be warned. Only the invalid data is removed from the form or field so the user can correct it. The system will keep the valid data in the form so the user does not have to retype it.

The user is unaffected by errors in associated system processes. Errors in an associated process of the system, such as a simulation, will not cause errors in another process, such as the user interface. Dependencies between processes must be kept to what is essential in order to control data flow, create user views and interface with the cluster controller.

c. System Feedback

The user can easily access valuable information from the system. The system will provide feedback when it successfully uploads files from the user, the user submits jobs to the cluster, the user creates a DOE, etc. The system will provide status of the simulation job processing queue. The system will alert the user when a job has completed (i.e., results are available).

2. Accessibility

Accessibility in this thesis is the availability of the system to the user in a variety of environments. The system will be available to the user when and where he or she needs it.

a. Remote Access

The user may access the system from any networked desktop computing setting. The system will not require the user to leave his or her work space to start a job, load files or retrieve results.

The user is able to start or stop simulation job processing and retrieve results from any networked desktop setting.

b. User Platform

The user can access the system without installing software or loading system plug-ins. Some settings within the Department of Defense make it extremely difficult to add non-standard software. System administrators often block new software for valid security and infrastructure concerns. They do not know if the software contains malicious code or how the software may affect other services within their area of concern.

The user can access the system from any modern operating system as long as the operating system possesses standard networking protocols. Simulations do not all work on the same operating systems so the system must work on at least Microsoft and Unix based operating systems. This will allow for widespread system adoption rather than pigeon-holing it into a small subset of users.

3. Security

User personal data, simulation model data and system use will be protected by the system. Simulations are used within the Department of Defense to make important strategic tactical decisions. If the system is openly available our adversaries can make assumptions on what our tactics might be from what we are studying. Simulations are also used in decisions regarding the purchase of billions of dollars worth of supplies and equipment every year. Advance views of the types of problems we are studying and the types of equipment we are simulating would give an unscrupulous manufacturer an unfair advantage in the contracting process.

a. Authenticate Users

The user can not access the system without registering. The system provides a user authentication scheme. The registration allows system administrators to track usage and allow or deny access to the site based on the user's credentials. Use of the high-performance computing cluster must be limited to qualified, validated persons.

b. Maintain Data Integrity

The user's data will not be available to other users, unless the owner requests it. The system must protect not only personal data, but also the models that the user loads and the results of simulation run. The system will provide collaborative abilities so a user can make his model's DOEs and results available to other users.

c. Protects user data and transactions

System use must be secure from unauthorized observation over the network. The system will obfuscate data and transactional requests sent over a network.

B. DEVELOPER ARCHITECTURE REQUIREMENTS

1. Modifiability

Administrators can quickly and cost-effectively alter data models, user views and business rules. It is not anticipated that this system will have a full-time developer or maintainer so there is a need for the code to be relatively accessible. Applying changes to the system will also require a minimum of time and effort.

2. Extensibility

One of the primary concerns with the software systems currently on the market within this problem domain is that they work great on only one simulation or with only specific designs. The system administrator will be able to extend the abilities of the system post-deployment.

a. Adding Design Algorithms

Administrators can add new designs and make them available to users within hours.

b. Adding Simulations

An administrator can make a new simulation available to the system within hours (not including loading the simulation on the high-performance computing cluster). The administrator must have some familiarity with the cluster in order to understand what data it requires to run a job. This function is restricted to administrators to maintain control over what is and is not available on a cluster and to maintain system security.

3. Maintainability

The administrator or developer can make changes to the system without creating a cascade in other portions of the system. A change in a function's interface will be the only change that affects other functions.

4. Portability

a. Server Operating System Dependency

The system's server operating system will not be determined by system requirements. Simulations and high-performance computing clusters run from a variety of operating systems. As such, the system will run on a variety of operating systems.

C. ACTIVITY DIAGRAM OF PROPOSED SYSTEM

A new activity model of the proposed application process flow was created based on the conclusions drawn in Chapter III. The model was then revised using input from members for the Simulation Experiments and Efficient Designs (SEED) Center for Data Farming, staff from the TRADOC Analysis Center Monterey, and Air Force Research Labs staff during presentations and discussions from March 2007 through July 20007. The final products (Figures 3 and 4) are the result of that feedback.

This section will highlight the difference between the original model, from Chapter III (Figures 1 and 2), and the final products below.

1. Design of Experiment Development

The two most prominent and important differences between the current DOE process (Figure 1) and the proposed process are the presence of the proposed system and the addition of two steps in the design point file creation process (Figure 2 top half). The details of these changes as well as any others are described below.

NOTE: the activity diagrams in Figures 3 and 4 have a non-standard notation. Some object flows are labeled with ':data' where 'data' is the attribute parameter that is being assign or updated in the flow. This was done to clarify the diagram.

a. Open Base Case File

This step was previously completed in the design point file creation process. This does actually add work to the DOE process, but will save time overall by combining steps and allowing the system to do work for the researcher.

This step assumes that the researcher has a working model that is verified to run on the simulation of choice for the experiment. This file is transferred through a network to the proposed system. The system will display a list of files that the researcher and his or her collaborators have uploaded. The BaseCaseFiles object (rectangle with underlined text) and associated object flow (dotted line) represents this action.

This step ends when the researcher selects the base case file that he or she wishes to use. This selection is recorded in the system.

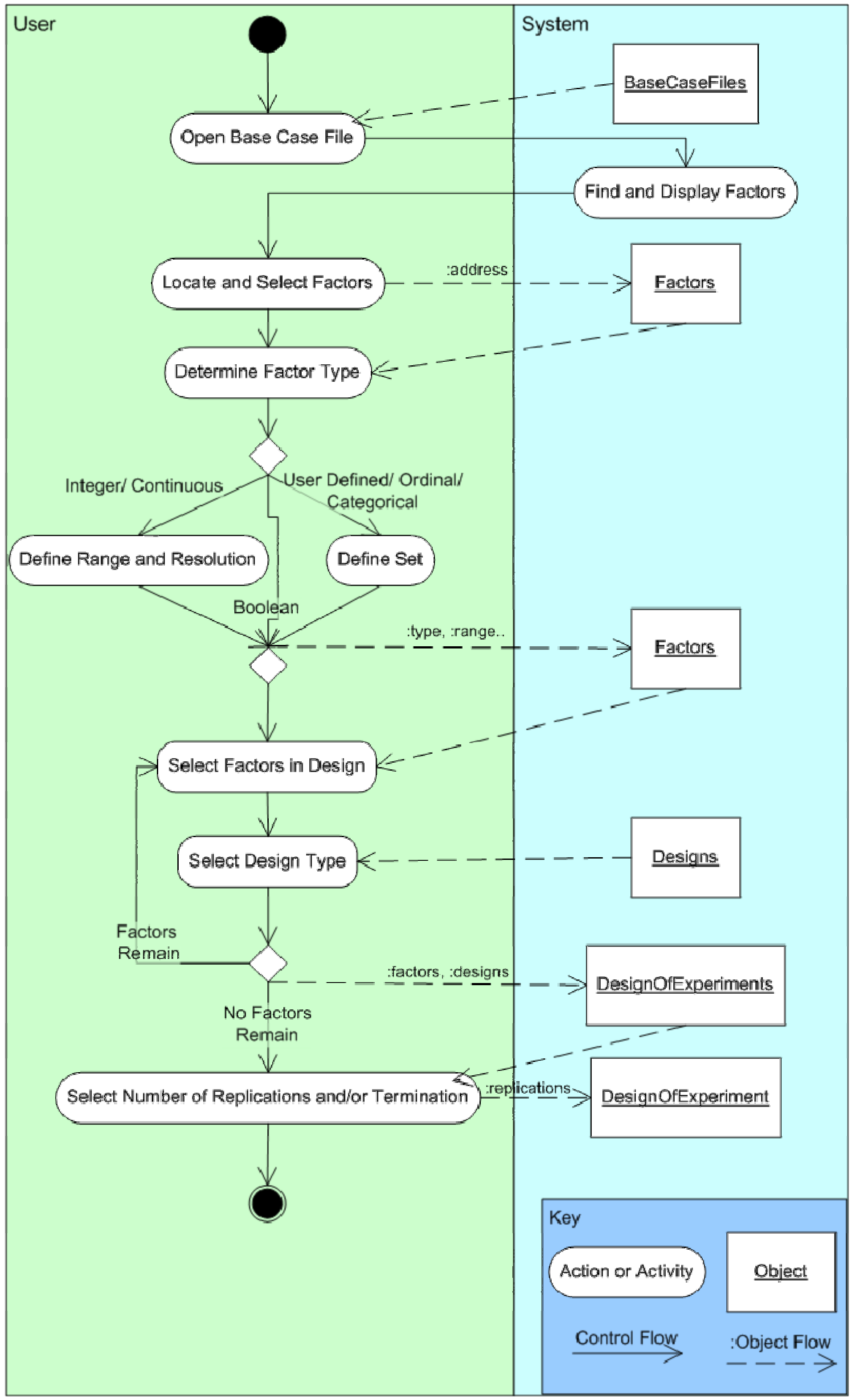

Figure 3. Proposed Design of Experiment Process

b. *Find and Display Factors*

This is the second step that was moved from the design point file creation process to the DOE process. In the current process the user does this by viewing the open text or database file. In the proposed system the work is done by the system. The system will process the base case file locating all possible factors. The system displays all factors to the user in a hierarchical tree format. This format represents factors as elements that are made up of attributes that describe the element and sub-elements, also called children, that are components of the element. An element, such as a car for example, may contain multiple attributes (e.g., color, number of doors) and possibly sub-elements (e.g., engine, suspension) which may have attributes and sub-elements of their own. Each element, child and attribute must be grouped with its parent in order to provide the researcher context. As described in Chapter III, a model file may contain tens to hundreds of thousands of factors. A flat, non-hierarchical, listing of the factors provides a confusing picture to the researcher. Preferably, the factors will be displayed in an expandable/collapsible list, like a Microsoft Explorer file structure representation, so the researcher does not have to view all of the children of elements that he or she knows contain no factors of interest.

This step ends when the factor and their relationships, derived from the base case file, are displayed for the user on a computer screen.

c. *Locate and Select Factors*

This step combines ‘Select Factors’ from the current DOE process (Figure 1) and ‘Locate Factor’ from the current design point file creation process (Figure 2). It is the final step that has been reallocated from the design point file creation process. As this step combines two steps from the prior process, it carries with it many of the inherent difficulties in conceptually and physically selecting the factors that were described in Chapter III. However, in combining these steps, then recording the user selections in the system, we take some of the cumbersomeness out of the process later on.

The researcher has selected the base case file and the system has found and displayed all possible factors. The researcher is now asked to select the factors of interest from the list of all factors. The coded location of the factor, in the Extensible Markup Language (XML) or database schema, is recorded in a Factor object for use in the next step.

d. Determine Factor Type; Define Range and Resolution; Define Set

The system has now collected the factor selections made by the researcher. The system clears the non-factors from the screen and displays the researcher's selections with the value and address from the base case file for reference. The remainder of these steps proceeds as previously described in Chapter III. The only difference here is that the decisions are made within the proposed system and the system records the decisions (as represented by the '*:type, :range'* object flow).

e. Select Factors in Design; Select Design Type; Select Number of Replications and/ or Termination

The remaining steps are the same as in the current DOE process except that they are completed in the proposed system. In Select Factors in Design the system provides the researcher with a list of the factors that he or she selected earlier in the process. In Select Design Type the system provides the researcher with the designs that have been encoded in the system. After all factors are allocated the system saves the factors and designs as a DesignOfExperiment object. Finally, the researcher enters the number of replications and/or the termination point for the simulation, which are also saved to the DesignOfExperiment object.

During these steps the system will provide the user with the number of design points that the completed DOE has. The product of the number of design points, the number of replications and the researchers estimated simulation run time (e.g., from a validation run) would yield an estimate of the total time that the experiment will take to process. The researcher can use this data to shape his DOE choices. Based on the time available and the estimated processing time the researcher can determine the resolution that he or she wants from the design. In a perfect world, with unlimited processing availability and time, this would not be a factor, but we must be practical about the situation.

f. Conclusion

This section ends with a complete DOE. In terms of time, the researcher is probably not ahead of where he or she would have been using the current process. However, unlike the current process, the information created during the proposed process is accumulated and will be directly applied during the next process. Also, unlike the current process the design decisions have been recorded and can be documented for retrospection and collaboration.

2. Design Point File Creation and Simulation Processing

The immediately identifiable difference between the current (Figure 2) and proposed (Figure 4) diagrams is that the proposed system has taken on the burden of the tedious work described in Chapter III.

This process starts with a DOE that was created in the previous process. It assumes that a base case file has already been saved to the system and the associated simulation software is loaded on the system.

a. Request Experiment Run

The researcher initiates the process by telling the system to run the DOE that was created in the previous process. The system takes over from this point until the results are generated. All other work prescribed for these steps in the previous chapter is automated by the proposed system.

b. Create Design Point Values

In the current process, the researcher completes this step using some external mechanism. The numbers are not used until we prepare to run the experiment, so there is no reason for the system to store the actual values until the researcher requests the experiment run.

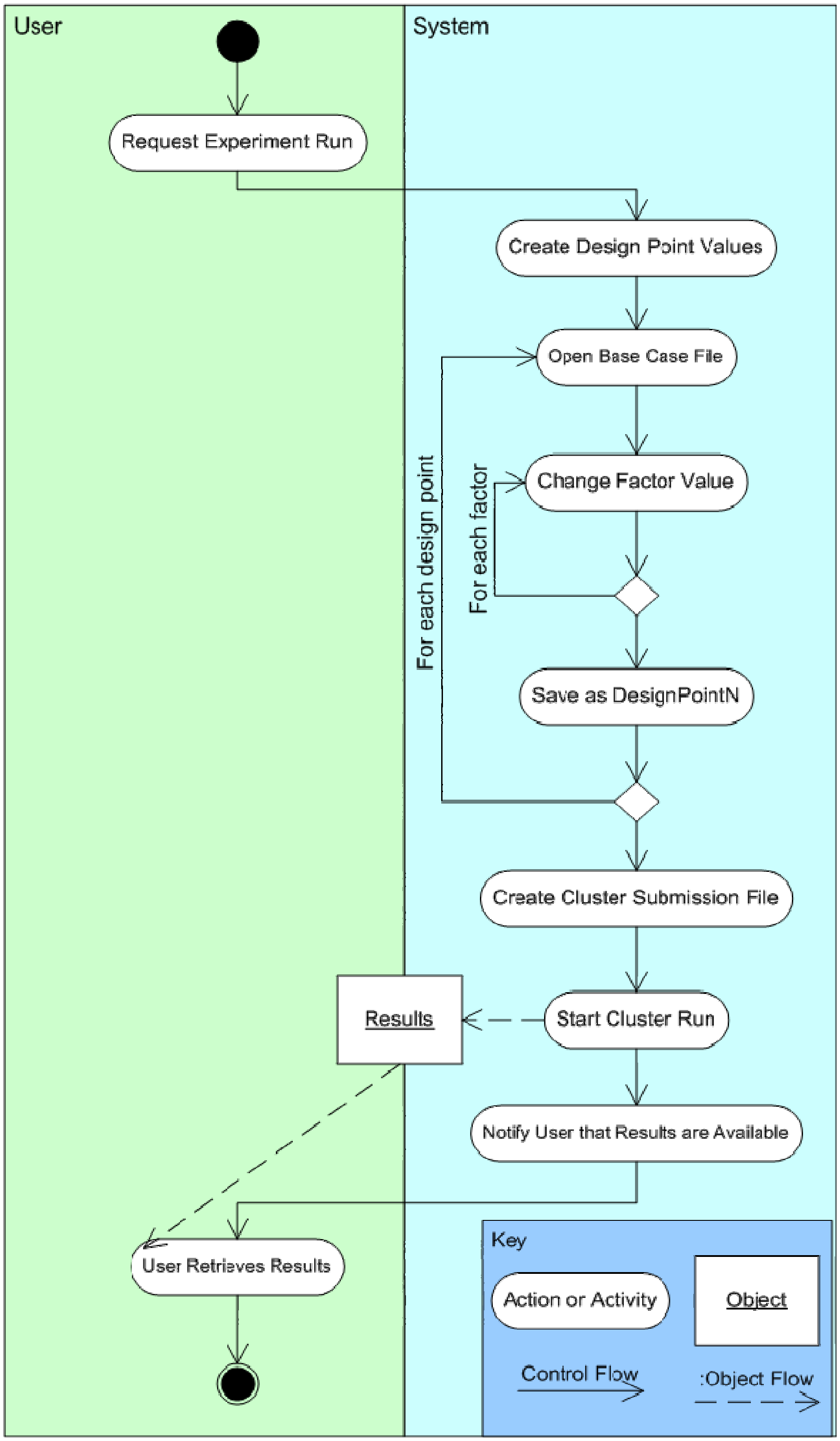

Figure 4. Proposed Design Point File and Simulation Processing

The system combines the information from the DOE object and the algorithm associated with the design to create the actual values for each design point. The values are stored in the system for application during the next step.

c. Open Base Case File; Change Factor Value; Save as DesignPointN

The proposed system opens a copy of the base case file that it renames based on the current design point, just as the user might in the current system. It then uses the factor address from the DOE to find the factors of interest. The system replaces each factor value with the corresponding value from the design point, created in the previous step. The system continues through the design point until it replaces all values for all factors of interest. The factor address may also allow the system to go directly to the factor of interest rather than scanning through the file as a human would. This gives the system another advantage, constant time access to factors, over a human processor who would have linear time access.

This group of steps is well suited to a computer. The process requires precision; a computer does not make mistakes unless programmed to. The process is repetitive and tedious; a computer does not get tired, bleary-eyed or get calls from its spouse about the kids misbehaving, distracting it from its task. If the system does have a higher priority task, it is smart enough to save exactly were it is, and then reload the information when this task is allocated processor time again.

d. Create Cluster Submission File; Start Cluster Run

The choice between running on a desktop or on a cluster is not available in the proposed system. A cluster controller can operate on a desktop, set to queue jobs to run only when the system is idle, but our interest is in high-performance computing clusters. The system will use the codified knowledge from the Cluster Subject Matter Expert to create submission files based on the requested simulation and the number of replications from the DOE file.

One additional consideration, especially for long-running simulations, is the method that the system uses to order design point simulation runs and replications. The first method runs all replications of a design point prior to moving on to another design point. This is simpler and only requires one cluster submission file. The second method starts each design point, prior to replicating any design point. Once the system submits all design points, it begins submitting the second replication of the DOE and so on. The advantage of this method is that if a research sponsor cuts the available time and the researcher needs to reduce the number of replications after they have started, he or she still has some sets of results for the complete DOE. A complete set of replications from only some of the design points is nearly useless. A complete set of design points is necessary to gain an unbiased picture of the response surface. Again, we see an advantage to using a computer for this process. Repeatedly making and running very similar submission files is a task well suited for a computer.

The second method was chosen for the proposed system. It is referred to as an *anytime algorithm.* An anytime algorithm can provide an answer at any time, but given more processing time the answer gains precision. These algorithms are important in Artificial Intelligence, but could be important to Modeling and Simulation also.

e. Notify User That Results are Available; User Retrieves Results

As results become available, the system will notify the user in order to avoid the package-tracking syndrome. Because parcel services have package tracking, people often compulsively check on the status several times a day, even if the website claims in bold-large-font lettering to only update the status once every twenty-four hours. To avoid this syndrome, the proposed system will email the user when results are available.

Finally, the researcher accesses the system and retrieves the results. The system should provide the results as archive files rather than hundreds of individual files.

D. DATA MODEL OF PROPOSED SYSTEM

Based on the requirements of the proposed system and the information requirements in the previous chapters, the data requirements were derived. An

Entity Relationship (ER) diagram shows the data grouped as entities and the entities' attributes. The remainder of this section provides an entity dictionary that describes each entity, attribute, and entity relationship. Rather than an alphabetical list, the dictionary lists the entity followed by its attributes then relationships.

Conventions:

- In the database table name is the plural form of the entity name such as *users*. An instance of the entity takes on the singular form, *user*.
- Each entity, except for base_case_files_users, has an <entity_name>_id primary key. The database will automatically assign this an integer value to uniquely identify each new entity instantiation and increment the value.
- Primary or foreign key designations precede an attribute definition.
- The data type (e.g. string, integer) follows each attribute.
- Relationships are annotated with a colon and the name of the related entity. For example, ':factors' defines a relationship between the entity being described and factors.

Entity: users - people who access the system including researchers and administrators

Attributes:

- first_name: the given name of the user; string
- last_name: the family name of the user; string
- email: the email address of the user; used to contact the user in cases where the cluster throws error or to alert the user when results are ready for download; string
- uid: the user selects a unique username; string
- hashed_password: the digested value of the user's password and the salt – a security measure; string
- salt: the randomly generated 'seed' that is combined with the password prior to digesting the password; string
- user_type: defines the user's level of privilege within the system, e.g. user, administrator; string

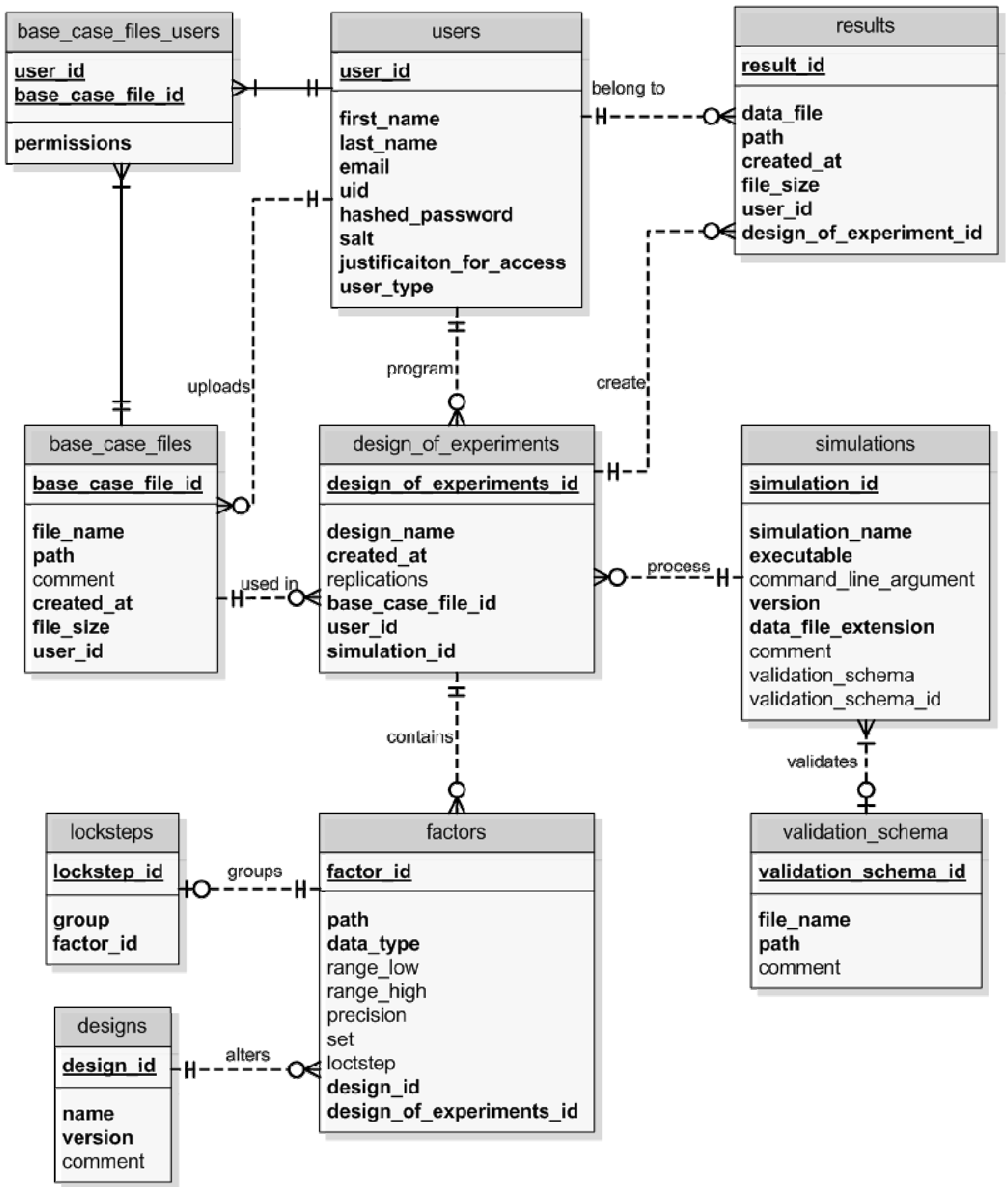

Figure 5. Entity Relationship Diagram of Proposed System

Relationships:

- :base_case_files: a user optionally uploads many base_case_files. A base case file must belong to a user
- :design_of_experiments: a user optionally creates design_of_experiments; a design_of_experiment must be owned by a user

- :results: a user optionally has many results through designs_of_experiments that are run; a result must belong a user
- :base_case_files_users: users are optionally assigned to many base_case_files_users; this relationship allows user collaboration through the sharing of base case files

Entity: design_of_experiments – theses entities are created by users to describe the method in which the factors will be manipulated

Attributes:

- design_name: required; assigned by the user to provide a descriptive identity to the design; not necessarily unique; string
- created at: required; the system records this required attribute to give the user a reference if he or she forgets the design_name; date/time
- replications: the number of times the simulation will run at each design point or the number of times it will run the base case file if that is the only file that the researcher wants to run; positive integer
- base_case_file_id: foreign key; required; identifies the file that the design_of_experiment will manipulate; integer
- user_id: foreign key; required; identifies the user that created this entity; integer
- simulation_id: foreign key; required; identifies the simulation that the design point files that this design_of_experiments will be applied to; integer

Relationships:

- :users: a design_of_experiment must be created by one user
- :results: when a simulation processes a design point file, created by the design_of_experiment, the end product is one or more results; a result must be created by a design_of_experiment
- :simulations: a simulation must process the design point file that the design_of_experiment creates
- :factors: a design_of_experiment optionally contains one or more factors.
- :base_case_files: one, and only one, base_case_file must be used in a design_of_experiment

Entity: base_case_files – users upload base_case_files that represent the base model that they intend to process with simulation software

Attributes:

- file_name: required; the name of the base_case_file that was uploaded to the system; string
- path: required; the relative path of the file in the file system; string
- comment: allows the user that uploads the base_case_file to further identify the file or leave a note for a collaborator that might be interested in the base_case_file; text
- created_at: required; auto assigned by the system; gives the user a frame of reference in case he or she forgets the name of the file they loaded; also allows administrators to identify old base_case_files to cleanse the system if necessary; date/time
- file_size: required; auto assigned by the system from file system information; provides the user with an idea of the disk size of the model; float
- user_id: foreign_key; required; identifies the user that uploaded the base_case_file; integer

Relationships:

- :users: a base_case_file must be uploaded by a user
- :design_of_experiments: a base_case_file can optionally be used in one or more design_of_experiments
- :base_case_files_users: base_case_files are optionally assigned to many base_case_files_users; this relationship allows user collaboration through the sharing of base_case_files

Entity: results – a simulation creates results; each result file contains the measures of interest that the researcher requested plus any simulation specific data; in this model, design_of_experiments provide the linkage between results and simulations

Attributes:

- data_file: required; the file name that contains the simulation results; string
- path: required; the relative path of the file in the file system; string
- created_at: required; auto assigned by the system; lets the user know how much they will be downloading; also allows administrators to identify old results to cleanse the system if necessary; date/time
- file_size: required; auto assigned by the system; lets the user know how much they will be downloading; string

- user_id: foreign key; required; identifies the user that the results belong to; integer
- design_of_experiment_id: foreign key; required; identifies the design_of_experiment that created the results; integer

Relationships:

- :users: a result must belong to one and only one user
- :design_of_experiments: a result must be created by one and only one design_of_experiment

Entity: base_case_files_users – this entity is an artifact of making base_case_files available to multiple users for collaborations

Attributes:

- user_id: primary key; required; identifies the user that is granted access to a base_case_file that another user created; integer
- base_case_file_id: primary key; required; identifies a file that is being made available to a user other than the user that created it; integer
- permission: defines the privilege level that the user has over the base_case_file; e.g. read, write; string

Relationships:

- :users: a base_case_files_user must be assigned one and only one user
- :base_case_files: a base_case_files_user must be assigned one and only one base_case_file

Entity: factors – simulation-element attributes that are of interest to the researcher

Attributes:

- path: required; the expression used to address the factor of interest in the data structure that it resides in; examples include SQL expressions for databases and XPaths for XML files; string
- data_type: required; identifies the class of data that the factor represents such as integer, float, categorical; string
- range_low: required for numeric data that will be varied along a range; represents the lowest value; float
- range_high: required for numeric data that will be varied along a range; represents the highest value; float
- precision: for numeric data, other than integers, this represents the number of digits to the left of the decimal point; integer

- set: for categorical and Boolean data; a comma separated set of strings representing the possible values for this factor; string
- lockstep_id: required; default value is no; identifies if the factor is to be changed as part of a group such as some characteristic of a weapon that needs to change for a squad of soldiers; Boolean
- design_id: integer; the design of experiment the will alter the factor to create design point files; integer
- design_of_experiments_id: required; the design_of_experiment that the factor belongs to; integer

Relationships:

- :designs: a factor must be altered by one and only one design, or else it isn't really a factor of interest
- :design_of_experiments: a factor must belong to one and only one design_of_experiment
- :lockstep: a factor is optionally involved in one and only one lockstep group

Entity: locksteps – identifies if the factor is to be changed as part of a group, such as a characteristic of a weapon that needs to change for a squad of soldiers

Attributes:

- group: required; a system assigned group number; there may be more than one group in a design_of_experiment; integer
- factor_id: required; the identifier of the factor in a lockstep group

Relationships:

- :factors: a lockstep belongs to one and only one factor

Entity: designs – a design contains the information necessary to alter a group of factors in a way that is consistent with some logical algorithm in order to achieve a statistically sound set of results to an experiment.

Attributes:

- name: required; a unique identifier for the design; it should be a recognized design such as NOLH or factorial; string
- version: an identifier for cases where the implementation of the design changes due to updates, errors etc. but the basic design remains; string
- comment: any additional information about the design, such as what types of experiments it works well for; text

Relationships:

- :factors: a design is optionally involved in altering one or more factors in any given design

Entity: simulations – software created to automate the processing of system models

Attributes:

- simulation_name: required; the name of the simulation software; string
- executable: required; the command required to start the simulation
- command_line_argument: any additional flags required when running the simulation; string
- version: required; the release of the software; some simulations are very specific about what they require in a simulation model and are not backward compatible; string
- data_file_extension: required; the filename extension for files that contain models for this simulation; this provides a first-line validation of the data file; string
- comment: any additional information of interest to the researcher or quirks of the simulation; text
- validation_schema_id: for XML files; a template that indicates the required element and attributes in a data model; integer

Relationships:

- :design_of_experiments: a simulation may optionally be involved in one or more design_of_experiments

Entity: validation_schema – a template that describes the required, optional and allowed data elements in an XML file

Attributes:

- file_name: required; the name of the file that will be test or validate model files prior to simulation processing; string
- path: required; the file system address for the validation_schema file; string
- comment: any additional information about the file that the system administrator finds useful; text

Relationships:

- :simulation: a validation schema must be related to one or more simulations

E. CONCLUSION

This chapter defined the attributes for the proposed system, mapped process flow through the proposed system, derived data requirements and modeled those requirements. The next chapter will discuss the development process of making these requirements into a prototype system.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DESIGN OF PROTOTYPE APPLICATION

The previous chapters described the current methods of Design of Experiments (DOE) and simulation processing, then described an architectural framework designed to improve these processes. The proposed system will automate steps that are tedious and error prone and will provide functions to apply the information input in the early steps of the process to where it is actually used.

This chapter will describe the decisions made in instantiating the architecture and developing the prototype. First, the choice of design methodologies is described. Next, we describe the design decisions such as choice of programming language and user interface component. Finally, the tools used in development are detailed. This will lead to the final substantive chapter, which will detail the development of the prototype system.

A. METHODOLOGY AND PATTERNS

1. Incremental Development

a. Justification

An incremental development allows the developer to plan delivery of completed functionality frequently, gaining stakeholder feedback; it reduces requirements risk and technology risks. Incremental development reduces requirements risk by frequently gaining user feedback regarding the delivered functions. This forces greater communication between the developer and stakeholders, allowing the developer to fix problems or adjust future development before the problems spread into other parts of the system. It reduces technological risk by allowing the developer to attempt to implement new or unproven technologies, serially or in parallel with other development, without affecting the implementation of well-proven technologies.

There is no known implementation of a system that aids in DOE and automates simulation execution so the requirements are not well known. The system must use some method to tie together preexisting technologies, such as cluster controllers and simulations. A relatively new scripting language, Ruby, does this well in many instances. While Ruby's first release was in 1995, it did not begin to gain traction until around 2000 with the release of the first popular English language tutorial [Thomas, 2000] and only gained widespread acceptance with the release of the web-framework, Ruby on Rails, in 2004. The immaturity of such a language means that many applications of the language have not been implemented, resulting in high technology risk. As mentioned, an incremental approach to design works well both of these situations.

Finally, the allotted time and development environment for this thesis suggests the use of an incremental approach is appropriate. Implementation of all functions mentioned in the previous chapters is unlikely within the time frame allowed. Rather than describing a system that only provides half of the functionality, this thesis plans for full functionality. Incremental development provides convenient break points where fully completed functions are delivered and the project is handed-off to a new developer.

b. Planned Increments

The system functions, as detailed in Chapter IV, section A, B, C and D, will be developed generally as follows: basic system framework; cluster and simulation interface; basic DOE; efficient DOE. Specific function implementation is below.

- remote access (basic user interface); upload files to server
- protect user transactions; authenticate users; maintain data integrity (no sharing); system feedback (interface with the cluster controller)
- adding simulations; request experiment run (base case file only); create cluster submission file with replications; start cluster run
- find and display factors; locate and select factors; change factor values (no actual design application); save design point file; run base case file and design point file
- notify user that results are available; user retrieves results

- adding design algorithms; change factor values and save design point files after design is applied
- determine factor type; define range and resolution; apply design to results; run simulation
- define set; select factors in design; select design type

2. Architectural Patterns

Architectural patterns are well-defined element and relationship sets that confer a known set of quality attributes and constraints, which aid in defining a system [Bass 2003]. They allow a developer to define in abstract terms the basic foundation of a system without dealing with low-level implementation details. For example, the developer of a file sharing system that needs the quality attribute of availability may say he will use a peer-to-peer pattern to provide the quality attribute prior to deciding on the actual protocols and procedure to implement that pattern.

a. Model-View-Controller

The Model-View-Controller (MVC) pattern separates the system data model, the user view of data and the control of the application. In this way, it achieves elements of usability, modifiability and maintainability quality attributes.

The model, in the MVC pattern, maintains all of the data as well as the state of the application. The model also enforces business rules as they apply to the data. For example, if the model declares that the number of replications in the DOE must be positive, the model ensures that a DOE object never contains a negative number for replications. In this manner, it also ensures that the database field for replications never has a negative number value (see ORM below).

Maintaining one data model aids greatly in the achieving maintainability quality attributes. In MVC, the developer must only look in one place to find and alter business rules about the data and the data storage schema. To ensure data integrity, a developer using some other pattern or proceeding haphazardly must search through all of the application code to find each function that might affect the data.

The view portion of the pattern is the representation of the model data that the user sees. The user never interacts directly with the data, only a view of the data. The application may display data in different ways for different functions. For example, a web application may display a different view for normal web browsers versus hand-held (e.g. mobile phone) browsers. The view is essentially dumb. It does not know what it is displaying or control which view is used, the controller does that. The view creates the display based on the data passed by the model. For all it knows a string passed to it could be a phone number or the works of Aristotle [Dempsey, 2003]. The view does provide a way for the user to interact with the application (e.g. form buttons, hyperlinks), but does not handle the function or request that the user makes.

The view portion of the triumvirate enables the developer to change the user interface rapidly, without worrying about disrupting the data model and causing a cascade of errors. This helps to achieve the modifiability and maintainability quality attributes. This also enhances overall usability by speeding up interface changes during the implementation and maintenance phases of development.

The final leg is the controller. The controller coordinates the application by acting as a bridge between the model and the view. It passes data and user input to the model. It tells the view what data to input as user feedback and selects the appropriate view based on the user instruction and user model. When the model rejects some data input that the controller passes it, like a negative number of replications that the user entered, the controller tells the view to redisplay the form with an angry message informing the user that you cannot replicate something negative times.

The model-view-controller pattern achieves several of the target system's quality attributes. It enables the attainment of maintenance and modifiability quality attributes by helping the developer write functions and rules only once. MVC enhances data integrity by keeping the data model separate from the application control and keeping the view separate from data manipulation.

b. Client-Server

Applying the client-server pattern achieves accessibility and security quality attributes. The client-server pattern can be seen as a two-tiered system with one system sending a request for a service and another system providing an appropriate response to the request.

In the case of the DOE system under development, the clients are remotely located from the server system. The users, on the client systems, will request views of the data model, which the server will provide as long as the parameters are within the defined controller logic.

B. IMPLEMENTATION DECISIONS

This section explains considerations in implementation of the architectural patterns described above to provide the quality attributes mentioned. Chapter IV will describe specific issues with instantiating these decisions.

1. User Interface

A web-based interface will provide remote access to the system. This provides open access to all networked computers that have a standard web browser. Development of the system with W3C compliant HTML rather than relying on proprietary plug-ins will ensure that the system is usable on all standard browsers.

A web-based system provides easy access, but brings security concerns. The private network at the Naval Postgraduate School is the initial target for deployment; however, broader implementation may include public network use. A transport layer security implementation, which encrypts client-server communications, will protect user data and user transactions from eavesdropping and tampering over both the private Naval Postgraduate School network and any future deployment on a public network.

A web-based system will also decrease new user training time by providing the user with a familiar interface. Web forms including drop down boxes and file upload dialogues are ubiquitous, and should be familiar to all users of the system.

A web-based system also enhances system modifiability. A client-server system, which depends on the installation of a standalone application for a client is prone to becoming outdated. In a web-based system the model, user view and controller logic reside on the server so the users always access the most up-to-date version of the application.

2. Language

As mentioned earlier, Ruby, an interpreted dynamic programming language, provides great utility for tying together various software elements. Ruby is open source with a liberal General Public License (GPL) so developers can use it freely, within scope. Ruby is also portable, one of our quality attributes, so it runs the same code on Unix, Windows, and any of over a dozen other operating systems. This means that implementing the system in a new environment does not require a developer to rewrite the code to work on that operating system. On the downside, interpreted languages are generally slower than compiled languages. If an organization wishes to deploy the system or release it for deployment in multiple locations, the developer should analyze any complex function for optimization or implementation in a compiled language if language speed turns out to be a bottleneck. However, given the relative speed of CPUs compared to network bandwidth, many web developers have reported that using interpreted scripting languages such as Perl or Ruby has a negligible effect on system performance.

The prototype system will use the Ruby on Rails web application framework. Ruby on Rails, or Rails, provides for easy development within the model-view-controller design pattern. Rails also allows rapid application development and testing. It is written in Ruby, which is not compiled, so no time is wasted waiting for the classes to compile after a small change is made. While you must restart most web servers to reflect any application changes, Rails' built-in web server, WEBrick, applies changes to the application on the fly. WEBrick does this by reloading the classes each time a user requests a page. This does make it is slow, but is very useful during development. The actual prototype deployment will use a different web server (see below) so the lack of speed does not detract from application utility.

Rails allows us to meet several of our quality attributes. Rails provides for modifiability, maintainability and usability quality attributes through its ease of use and developer tools such as form validation. Rails handles many low level details that other web frameworks do not. For example, configuring Rails to work with most any database only requires the installation of a RubyGem, which takes one entry in the operating system's command line environment.

Rails provides Object-Relational Mapping (ORM) through its Active Record class. ORM maps scalar data from a relational database to objects (i.e. object oriented programming objects). A database can store volumes of data, but not objects. In addition to data, objects can hold state and have behaviors associated with them. They can also implement inheritance from parent classes. Unfortunately, we cannot just work with objects in the server's primary memory (RAM). The data available to most web applications exceeds RAM for most servers, so databases are handy in this regard. Active Record handles all of the mapping tasks. Tables are mapped to classes; rows to objects. Active Record's automated mapping takes care of this detail, which the developer using many other frameworks would have to handle on their own.

Rails' tight ORM integration and database adapters, which hide the details of application-database communications, provides for easy database migrations and easy data manipulation. A database migration changes the database schema and can also insert data into the database. Rather than learning different commands for each database brand, Rails provides one programming interface, then handles the actual database manipulation for the programmer. Therefore, if the database software available to the organization changes in the middle of development, web development does not have to restart from scratch with coding the new data manipulation commands.

Using Rails makes application usability easier to achieve also. For example, Rails includes form validation helper methods. Within the model classes, the developer can implement these powerful methods with one line of code. If we need to ensure that the user enters a number in the 'Replications' field, we need only enter the following in the design_of_experiments model:

```
validates_numericality_of :replications
```

Now, because of this code and because Rails uses the MVC framework, on *any* page where a user enters replications Rails will check to see if the entry is a number prior to saving the value. If the validation fails Rails redisplays the page retaining all the values that the user previously entered plus a message at the top of the page telling the user of his or her erroneous entry. To override the default message, the developer adds an option:

```
validates_numericality_of :replications
                  :message => "Must be a number."
```

To validate multiple fields in a form with the same method the developer need only list them out with a comma between each:

```
validates_presence_of :first_name, :last_name
```

These snippets illustrate the ease of implementing helpful methods within the Rails web application framework. They should also show how this would enable the developer to rapidly create and rework a program within this framework.

3. Servers

A web-based application creates Hypertext Markup Language (HTML) documents on the fly (i.e., dynamically) by combining templates and information from a database based on the user request. Ruby on Rails provides this function, but it does not handle the receipt of data requests and sending of the dynamically generated HTML document back to the user. A web server does. Two applications will serve web documents for the prototype system. Mongrel is a fast, easy to set up, stable web server, but is single threaded, which means that it could not handle multiple requests simultaneously (i.e. only a couple users at a time). This is overcome using lighttpd, which does not handle dynamic web pages as well as Mongrel, but can serve up static content much faster and, more importantly, can balance requests between multiple servers.

To take advantage of these capabilities, multiple Mongrel servers run behind a lighttpd server. Lighttpd is the only portion of the set up that faces the internet. As lighttpd receives requests from the internet, it 'decides' which Mongrel server it should hand off the request to. The decision is configurable in a number of ways, but inconsequential here. The Mongrel server then retrieves the content and passes it back to the lighttpd server that sends it along to the requester.

4. Computing Cluster and Cluster Controller

Implementing the cluster is beyond the scope of this thesis, but the constraints that the prototype was built under should be documented. The high-performance computing cluster (HPC) that this research sought to increase utilization of was transferred from the Maui High Performance Computing Center (MHPCC) in Hawaii. The cluster consists of 12 desktop computers with dual Intel processors and 2 MB of RAM running Windows XP. It provides an overall processing performance of over 20Tflop/s.

A cluster controller is a software application that manages and exploits the available processors as effectively as possible. This software accepts job submissions, queues them to run, prioritizes resource allocation, monitors the jobs' status, provides feedback, then informs the user upon completion [Condor 2007]. One of the research sponsors for this thesis selected the Condor High Throughput Computing software for the cluster controller and was using it prior to the start of this research.

5. Database

As mentioned in the Language section above, the database that Rails interfaces with is of little consequence as long as it belongs to the group of a dozen or so that Rails has adapters for. MySQL [MySQL, 2007] is an open source General Public License (GPL) relational database management system in common use in the U.S. and abroad. MySQL was selected as a matter of convenience and cost. It provides all the basic functions of a database and because it is widely used, there are many user forums in case of problems. The GPL license means that there was no cost to implement the database for research purposes.

C. DEVELOPMENT TOOLS

The development tools are described mainly because they can have some impact on the resulting application and code. For example, an application developed in Sun's Netbeans [2007] integrated development environment (IDE) uses Swing class object for graphical user interfaces (GUI), while the Eclipse IDE [2006] uses the SWT package. This difference can result in incompatibilities.

The development environments used for this project are detailed below. Neither environment is meant to portray the minimum system requirements for the prototype. Rather, the environment information provides the reader a context in which to recognize any system specific technicalities in the following chapters.

1. Eclipse, Subclipse

An integrated development environment (IDE) is a software tool that aids in developing other software. There is normally a graphic user interface to give the user access to common tools (by mouse click or keyboard shortcut) and a file browser so the developer can easily access all of the files in an application. Some common tools are debuggers, to find program errors, and integrated compilers to turn human readable code into machine or byte code. IDEs also contextually highlight reserved words and variables and can automatically format code so it is easier to read.

Ruby on Rails' basic file structure contains thirteen top-level folders plus several sub-folders. Each time a developer adds a model Rails adds six files (i.e., the model, view, controller, and test files for each), so moving back and forth throughout the file structure would be distracting from a file navigation window and difficult from the command prompt. The IDE handled these issues transparently.

The Eclipse Platform [Eclipse, 2006] was selected to aid in development during this research. Eclipse, originally developed for Java programming, is a framework in which developers can easily design plug-ins to aid in the development of any programming language.

To aid in Ruby development, an open source plug-in named Ruby Development Tools (RDT) was added to Eclipse. RadRails [RadRails, 2006] provided Rails development functionality such as running migrations and starting the web server from within the Eclipse platform.

In order to preserve the revision history, back-up code, and make it available from multiple locations, a versioning tool was used. A Subversion repository was available for this function. An additional plug-in, Subclipse [Subclipse, 2007], provided a Subversion interface from within Eclipse.

2. Development Platform

Initially development took place on a Dell notebook computer, but due to a hardware failure the development was switched to an Apple Powerbook G4. Because all of the code was backed up with Subversion, and since Ruby and Rails are not operating system specific, the transfer was seamless. The author chose Ruby because of its portability and the value of this choice was inadvertently demonstrated by the hardware failure.

Later in the development, the Dell was fixed. Subclipse synchronized the codebase on the Dell with that on the Mac. A database migration was then run from Eclipse to bring the database schema up to date. After the database migration, the remaining system components were started and the system perfectly mirrored the system running on the Mac.

System specifics:

Dell Inspiron E1505 with a Intel T2400 Core Duo CPU running at 1.83 GHz and addressing 1 GB RAM. The operating system was Microsoft Windows XP Media Center Edition, Service Pack 2

Apple Mac PowerBook G4 with a Power PC G4 processor running at 1.5 GHz with 512 MB of RAM. The Mac operating system was Mac OSX Version 10.4.10.

VI. PROTOTYPE IMPLEMETATION

A. INCREMENT ONE (SYSTEM BACKBONE)

The goal of the first development increment was to provide a base upon which the rest of the system would be built and to provide some simple functionality to test the base. To accomplish this goal the web server and database were installed and configured, then a skeleton website was created with a function to upload files.

1. Remote Access

A web server enables remote access to a web based application as described in Chapter V. Lighttpd, commonly referred to as ‘lighty’, and Mongrel web servers were installed with lighttpd facing the user and backing to a cluster of four Mongrel web servers that would serve up the dynamic web content. The installation was performed according to the instructions on the Mongrel website [Mongrel 2007] and was uneventful.

Ruby on Rails was already installed on the development platform so the author created a new Rails project named ‘RESIDE’ for REmote SImulation and Design of Experiment. A model file representing users was programmed, and then a scaffolding for viewing the users model in the website was created. RESIDE was opened in a web browser and it displayed properly. A test user was successfully added.

Once the application was running, the server load testing could commence. Httperf [Mossberger & Jin, 1998], a tool that measures web server performance by generating HTTP workloads and providing reply status as output, was used to bombard the servers with various amounts of ‘traffic.’ Httperf ‘hit’ the server with up to ten consecutive connections for three thousand total connections resulting in three-thousand replies with about five replies per second. All replies were 2xx replies, which means a connection to the website was made. 3xx series replies indicate a redirection and 4xx (e.g. 401) are errors. All test results were saved to a log file. Overall, the setup worked well.

2. Upload Files to Server

This increment proceeded with the creation of the base_case_files model to represent files uploaded by users. While Rails does inherently allow for uploading files it loads them into the database. This method results in a degradation overall application performance [Attkinson, 2001] so a plug-in was found to work around the problem. The plug-in, Acts As Attachment, provides a configurable way to upload files from a system user and save them in the file system. The plug-in developer created Acts As Attachment to upload picture files, so a few of the default setting needed to be changed when the plug-in was implemented. The maximum file size, which was one megabyte, was increased to ten megabytes to accommodate large simulation models. Acts As Attachment also required a slightly different database schema for the target model to work correctly. The 'file_name' attribute from the base_case_files table had to be changed to 'filename', and three attributes were added. The system does not use the new attributes, content_type, parent_id and thumbnail, so they remain as artifacts of the plug-in.

After model creation and configuration, a scaffold view was built to test the upload function. Uploading worked well so another view was created to display the record with a link to download the file. Downloading the file worked.

An overall website design was created. The design gave the user the ability to view all of his or her relevant information in one page (Figure 6).

The left side of the page provides navigation links. An optional user feedback section displays at the top of the main section. This section displays messages such as "You are not authorized to view that page." or "File successfully added." The message section, a *flash message* in Rails parlance, does not show up if there is no current message.

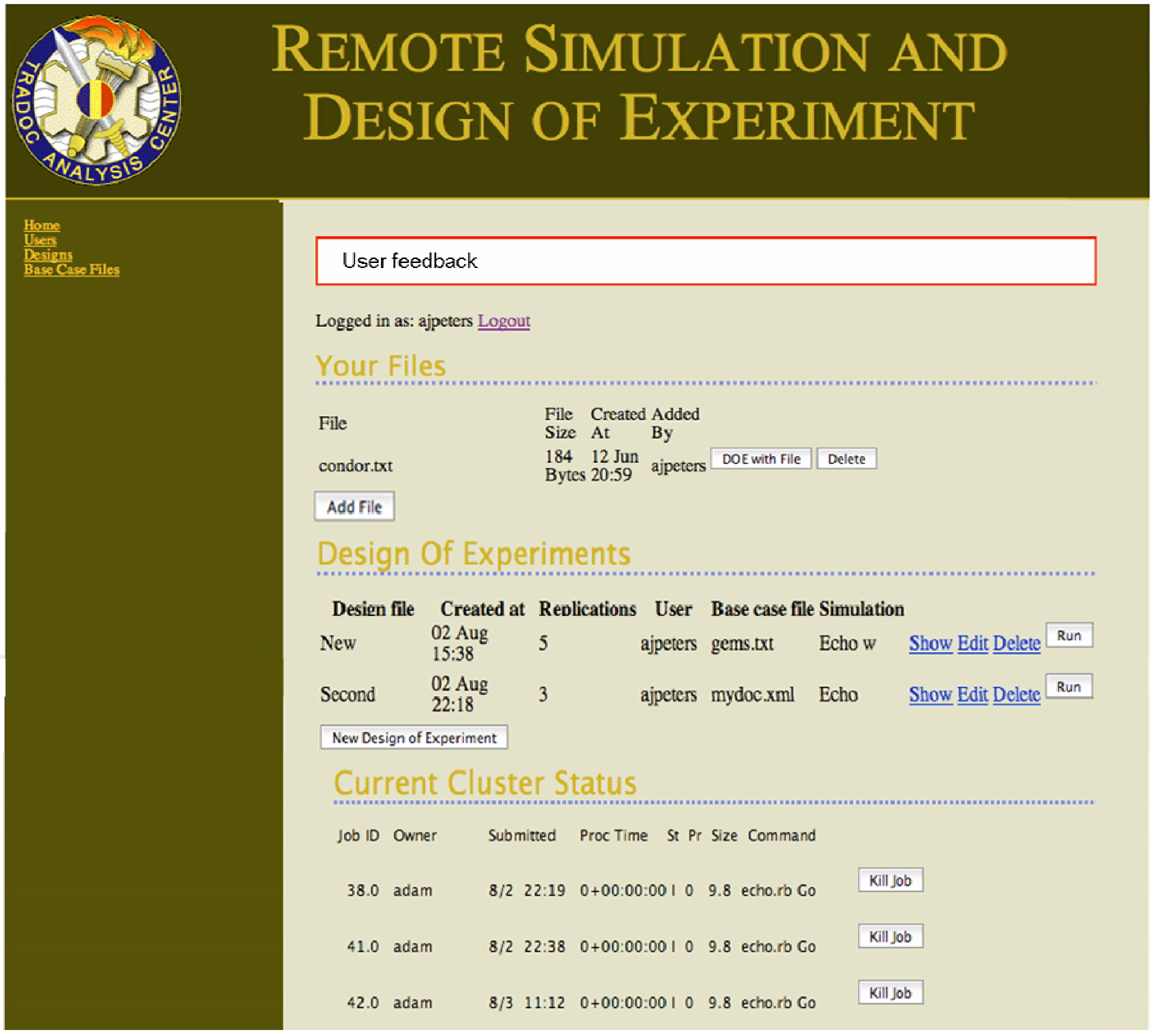

Figure 6. Main Application Page

The next section lists any files that the user uploads to the system. A file is not visible to any user other than the one who uploaded the file (i.e. the owner). However, the owner can share the file with others[1]. This section has buttons to add a new file, delete a file or create a Design of Experiment (DOE) with the file. If a DOE contains a file from this section, then the user cannot delete the file. The system displays a message indicating the user must first delete the file before deleting the DOE.

B. INCREMENT TWO (SECURITY AND CLUSTER INTEGRATION)

Security was added to the system and the cluster-controller was integrated into the system prior to adding additional DOE or simulation run

[1] Not implemented yet.

functions. The security at this level will encrypt the user's passwords, the files that they upload or download, and the DOEs that they make. The cluster controller will then be added and will inform the system user as to the current activity on the cluster.

1. Protect User Transactions

As mentioned earlier, encryption will hide user data transactions including authentication. Accomplish this was straightforward. First, a server certificate was created. In the development environment, a 'self-signed' certificate was used. In deployment, the system administrator should have a 'trusted source' sign the certificate. Next a second lighttpd configuration file, lightypd_ssl.conf was created to handle https: requests. The new configuration file includes the certificate location, so order of evaluation matters. Finally, one line is added to the application control file. All views in RESIDE were made to require secure communications, but changing this for some views only requires a small modification to the code. If a user enters an http: URL prefix rather than an https prefix, Rails automatically redirects the communication to https.

2. Authenticate Users

User authorization requires the implementation of filters within the controller objects. A filter is a function that can be set to run before, after, or before and after a given set of code executes. In this case, a 'before filter' checks to see if the user has authenticated prior to accessing a restricted portion of the application. To authorize, the user enters his or her user name and password. If a user does try to get to a restricted portion of the website without signing in, the filter catches it and redirects the user to the login page. The system remembers the original page request and sends the user to that page if he or she can supply a valid username/password pair.

After the authentication filters and functions were complete, a function was implemented to grant privileges within the system. For example, a system administrator should be able to add new simulations, while a casual user should not. The system of filters provides these constraints. They check the user's user_type attribute and display the requested information is the user is an administrator. If the user is not privileged, the system returns the user to where he or she came from with a message added to the top of the screen indicating that the user does not have proper privilege to access that page.

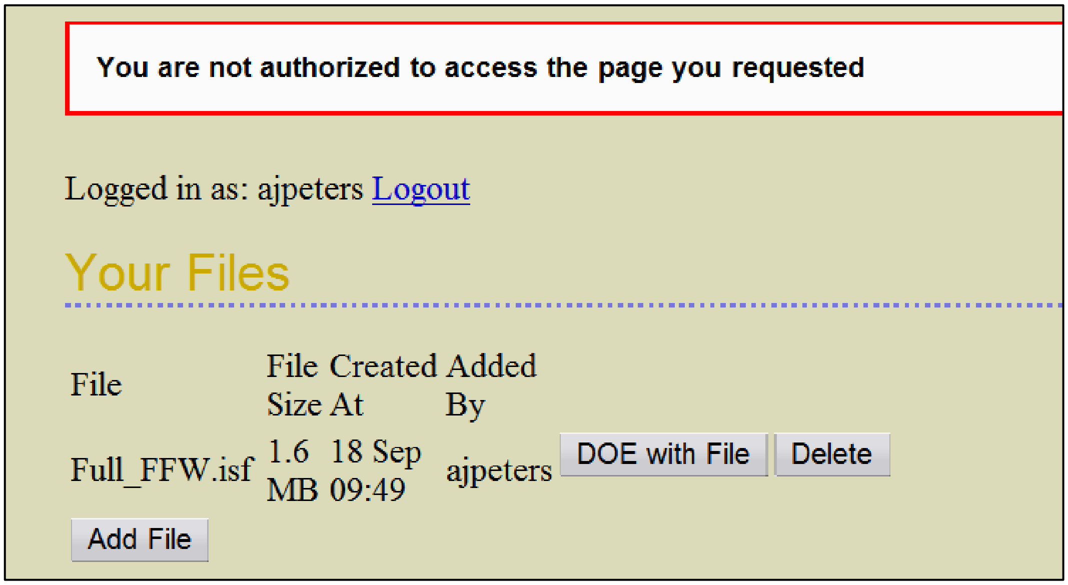

Figure 7. User Denied Access to Add Simulations

3. Cluster Integration and System Feedback

The Condor cluster controller provides the user feedback on the status of the high-performance computing cluster (HPC) when ordered to in the command line environment. It provides information such as number of jobs, how long those jobs have run and the job's creator.

Because RESIDE system users will not have command line environment access to the system and because the purpose of this research was to abstract such specialized knowledge requirements away from the user, this function was added to the web based system. The bottom of Figure 6 shows the results. A button to remove a job from the queue was added later in development. This function currently relies on user benevolence as it allows the user to kill both his or her jobs as well as other's jobs, since all jobs are owned by the condor system and all users need access to the system run-control functionality.

C. INCREMENT THREE (SIMULATION INTEGRATION)

This increment added the first target function to the system, running a simulation on a remotely located HPC. To accomplish this, a function to add simulations to the system was created. Then, the system was programmed to create the submission file, which the cluster controller would use to run the simulation with the indicated base case file.

1. Adding Simulations

This task proceeded with no significant issues. A new model was added to the system and the corresponding controller and view files were programmed. As mentioned earlier, adding a simulation is a privileged function so a "before" filter provided authorization to these functions. The view creates a basic web form. If the model validation function finds no problems, then the user sees a screen like Figure 8.

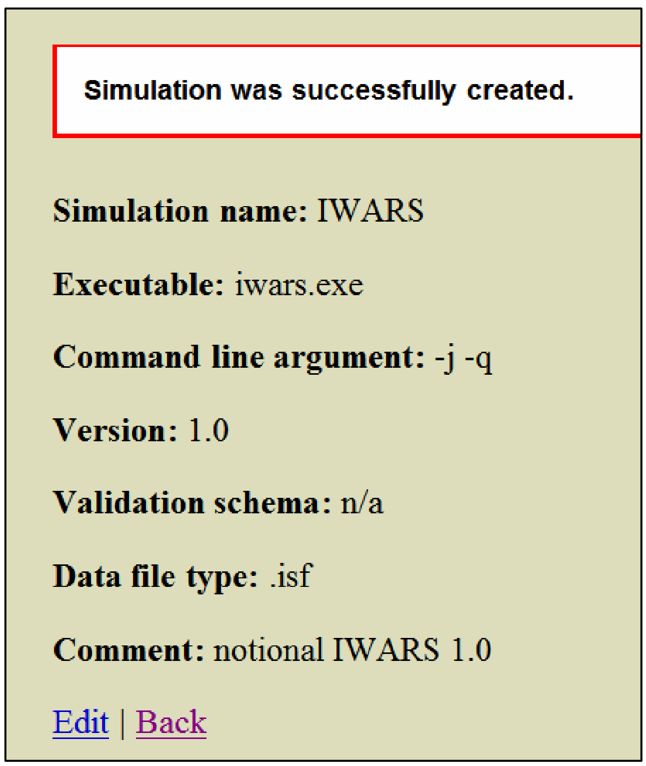

Figure 8. Simulation Creation Feedback

Of note, Rails offers a 'layout' function that performs like a template for system views. Figure 8 is cropped, but if it were not, it would show the same layout as Figure 6. The main section of the screen contains the simulation form, the remainder is the same. This was done by creating the 'common' layout in the layouts folder then programming the Simulation Controller object with the "layout ‘common’” command. The common layout then renders everything up to where the developer tells it to yield to view specific information.

2. Request Experiment Run

To achieve the remote simulation run function and stay within the system framework the user must first create a DOE. The DOE, at this stage, only allows the user to select the desired number of replications for the simulation

As Figure 6 shows, there are two methods to create a DOE. The user can select a button next to the base case file or can select the “New Design of Experiment” button within the Design of Experiments section. If the user selects the button next to the file, the system will fill in the “Base Case File” field of the New Design of Experiment form. Otherwise, the system provides a drop down box that the users use to select the base case file. To proceed, the user enters a design name of his or her choosing and the number of simulation replications. Finally, the user selects the simulation software that will process the model from the bottom drop-down box (Figure 9).

Currently the system displays all available simulations in the drop-down box. Future development should display only simulation packages that are compatible with the type of Base Case File, as indicated by its filename extension (e.g., ‘.isf’ in the figure).

Now that a user can create a DOE, the function to create the cluster submission is added. The system is first programmed with a generic template that satisfies the requirements of the cluster controller. The system then inserts the values that it derives from the DOE such as the number replications and base case file. The DOE also specifies the simulation to use. The simulation object data are then used to populate the submission file with variables such as the location of the executable simulation program.

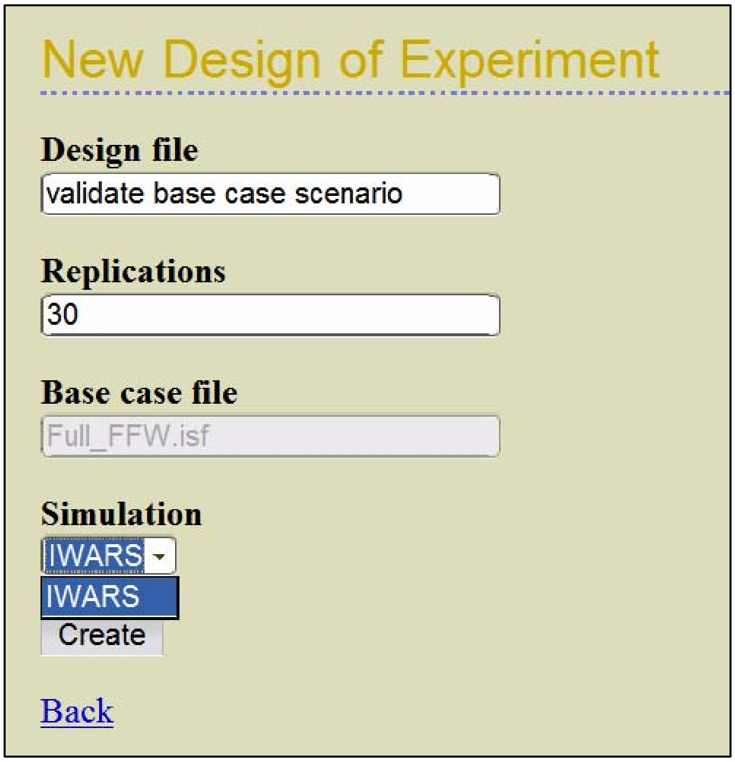

Figure 9. New Design of Experiment with Base Case File Pre-Selected

This increment concludes with the addition of a system function to start the cluster run. The system was programmed with the command to submit the file that was just created to the Condor controller, then put Condor's reply, "N job(s) submitted to cluster #X." in the flash message portion of the user screen.

D. INCREMENT FOUR (START TRUE DESIGN OF EXPERIMENT)

The final increment that this thesis recounts will allow the user to select a base case file for a DOE then find and display the factors in that file. After the system displays the factors, the user should be able to select the factors that he or she wants to include in the DOE and the system should record those selections, alter the numeric factors according to the selected DOE → factor mapping, then run a simulation with both the original base case file and one with the factors altered.

1. Find and Display Factors

This concept was the biggest technological hurdle in the research. The system needed to open a data file, find all the factors within that file and display them for the user in some meaningful way. In most cases, the data that represents a simulation model is either in a database file or in an Extensible Markup Language (XML) file.

The approach to parsing these files is similar, but the actual implementation is very different. In both files types, the top-level structures are first exposed, then recursively broken down until the program decomposes the data down to its atomic components. In databases, the top-level structures are exposed by examining the data dictionaries, which list the tables. Inspection of each table provides the attribute names that describe the row values. Finally, the program fetches each row value and recomposes the data for the user. The actual display of the data is discussed below. The difficulty with database files is that each implementation requires different adapters and slightly different coding to decompose. While it is more of a caution for the system user than a development issue, another danger with relational databases is that they often have integer “key” values that point to records in other tables. If the references are mistaken for factors and the values are altered in the DOE, the simulation results will be useless.

A root node or element is an XML file’s top-level structure in XPath, the language for finding information in an XML document. Every element in an XML file can have a value, a child node, one or more attributes, or all three[2]. Every child node in an element is itself an element, so it can have children, attributes and values also. Decomposing an XML document requires that, for each element, the program list the element value, then all attribute names and values, then the program must parse each child in kind. Every XML file may have a

[2] There are actually seven element node types in XPath, but the others are unimportant to this discussion.

different structure, based on the number and types of attributes and child elements, but every XML file is text[3] so the programmer can handle each in the same fashion.

The development goal for this thesis was to create a proof of concept system, so only one type of model file, XML, was parsed.

As discussed in earlier chapters each model may have thousands of data values. The proper display of the values is a major usability issue for this system. The first part of the issue is what to show the user, the second part is how to show it.

The question of what to show the user revolves around the balance between being flexible and being usable. To provide the most flexibility the system will display every node to the user. Within these nodes, there will be great deal of chaff to sift through to find the wheat, but the wheat will be there. To provide the most usability an expert user can prescreen the data file schemas for each simulation to find the node types that have farmable[4] factors. At run-time the system will only display the selected factors to the user. The problem with method is that the wheat might not be there when a researcher with a different point of view tries to design an experiment. As discussed in Chapter II this was the case with the Tiller.

The question of how to show the data in a usable fashion is easy to answer, but was difficult to execute. The data are naturally nested, either in tables and rows in a database or in nodes and child nodes in XML, so an expandable/collapsible list, much like a folder and file listing in Microsoft Explorer, was selected as a means to provide more manageable factor display for the user. With some knowledge of the model the user should be able to skip over hundreds or thousands of lines of potential factors by not expanding a node where he or she knows there is nothing of interest. For example, if the

[3] There are binary XML representations, but they are not in common use within the Modeling and Simulation domain.

[4] Large scale exploratory DOE has come to be known as “data farming”, and farmable factors are ones which are suitable for exploration in this context.

experiment deals only with evaluating blue force equipment packages, the nodes that represent the red and green forces can be left collapsed, saving the researcher time.

A JavaScript package, aqtree2 [Aqtree2, 2002], and its associated cascading style sheet file provides the ability to render expandable/collapsible lists without installing any plug-ins on the remote user's browser. Almost no DoD command allows users to install plug-ins due to security concerns, but JavaScript is still active on most browsers so aqtree2 is a viable alternative. The aqtree2 script makes the user's browser render well-formed unordered Hypertext Markup Language (HTML) lists as expandable/collapsible lists.

Figure 10 shows two states of the same web page, one collapsed to the top level, on the left, and one partially expanded. Prior to transformation and rendering, the source data file was ten thousand five hundred lines long, but it is displayed here in a dozen or so lines.

The function that transforms the XML files 'hides' the file location address for each factor in the HTML. As the user works through the model, selecting factors of interest the '-'s turn to '+'s as with 'Squad Active' and 'NumAgents' in the figure. When factor selection is complete, the user submits the form and the system records each selection and the selection's address in the Factors table.

E. CONCLUSION

The effect of rendering the expandable/collapsible list was difficult for the author to achieve. It eventually broke the development timeline and ended the current research effort. The code to process an XML file into a list has three remaining issues. These are HTML file optimization, transformation processing speed, and rendering speed.

A snippet of JavaScript is embedded in every line to turn the '-' to a "+" and vice versa. This adds several thousand lines of HTML to the resultant file. Embedding the code in the header once should fix this issue.

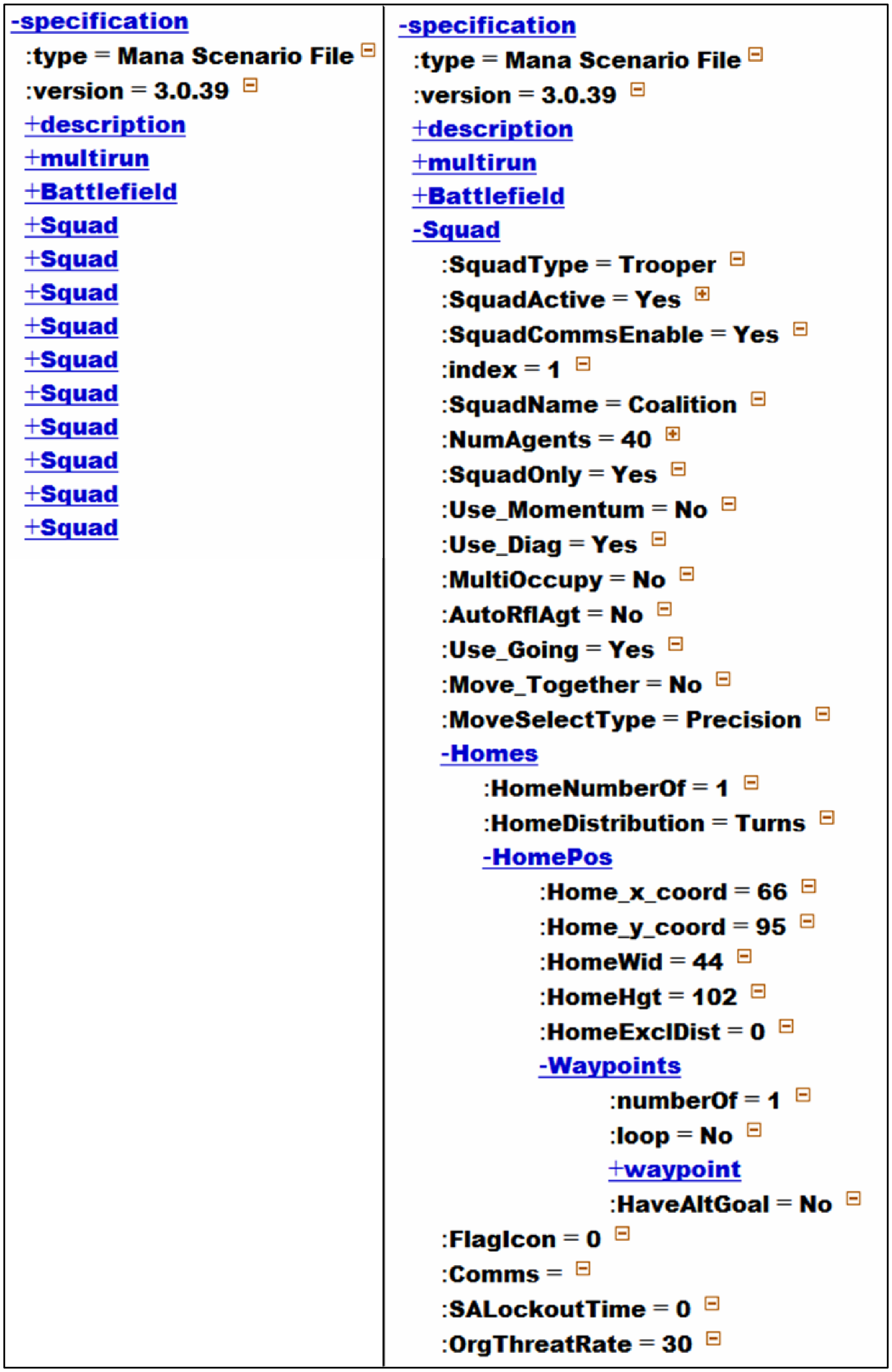

Figure 10. Expandable/ Collapsible List

The second issue is transformation time. Transforming a one-hundred-line file takes a few seconds. A ten-thousand-line file takes a few minutes. A six-megabyte file took about seven hours to process from XML to HTML. Using a compiled language to do the transformation may help the speed dramatically. Optimistically transforming the files would also decrease user wait time. That is, as soon as a user uploads a file, the system should start processing the transformation. This may waste some processor time, but user time is more important.

The last issue is the time it takes the browser to render the expandable/ collapsible list once the list loads. The browser, with the JavaScript helper, must examine the structure of the list to find which elements are parents, and which children belong to which parent. Because the data model is extensive and can run many layers deep, it requires a great deal of processing power and memory. The file in Figure 10 loaded into the browser in just a few seconds; however, it took about four minutes to collapse, during which time a user cannot do anything. The two most likely resolutions here are to not do the expand/collapse bit, or to just leave it as is. In the first method, the user will have to comb through the factors, but only one time per DOE rather than one time for each design point. In the second method we notify the user that the operation is going to take some time, and tell him that he should get a soda, check his notes, and then come back to make the selections. In either case, the user is still ahead, if we can apply the algorithmic design to the resultant factor selections.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND FUTURE RESEARCH

A. SUMMARY

This thesis described the need for a reusable architecture for automated Design of Experiments (DOE), design point file creation, and for running simulations on a remote high-performance computer cluster. Then it detailed the important quality attributes of that architecture. An activity diagram of the proposed system aided in extracting the specific functional requirements for the architecture. Next, the research derived the data requirements from the quality attributes and the functional requirements flow. After detailing all of the requirements, the thesis described and justified the selection of specific components necessary to implement the architecture in a prototype system. Finally, the thesis described the implementation of the architecture in a prototype.

B. CONCLUSIONS

The trend in Modeling and Simulation software both in the military and civilian domains is toward tighter overall coupling of functions, such as simulation software that includes drag and drop model development or modeling software that includes some DOE function. This trend is distressing. Organizations will end up purchasing or developing the same function, such as model building, repeatedly for each simulation software package that they purchase.

Although the focus of this research was the design and implementation of a system to complete DOEs and run experiments on remote high-performance computing clusters, its most important finding is that tools can be developed that will work with a variety of simulation systems as long as those systems provide open interfaces and act in a standard manner.

In order to curb the trends mentioned above and to create reusable tools the Defense Modeling and Simulation Office (DMSO) was recently renamed the Modeling and Simulation Coordination Office (M&S CO) [MSAIC, 2007]. The intent behind the move is to signal the change from stovepipe modeling and simulation efforts segregated by commands or systems to a more integrated approach in which reuse of simulations, tools, and services across functional domains is common. This is a bold move forward, however DMSO was intended to reach similar goals both when they were initially constituted and when they were ‘redirected’ by the Modeling and Simulation Master Plan in 1995 [DODD5000.50-P, 1995].

C. FUTURE RESEARCH

Several more increments in the prototype development still remain. Each increment is challenging and provides a ripe area for research. In addition to moving forward, two system functions require optimization to make the system more usable. They are the speed of creation of the expandable/collapsible lists of from base case files, and the speed at which a browser displays the lists. To increase the speed of list creation, the researcher may wish to implement the function in a compiled language or the much faster - but less functional and just as poorly documented - Libxml-Ruby [Libxml, 2006], the Ruby language binding of the GNOME Libxml2 library which is programmed in C.

To improve display speed the future researcher may find an Asynchronous Java Script and Extensible Markup Language (AJAX) method such as Live Tree [Live Tree] useful. Live Tree loads only the requested parts of the list. Initially it only loads the top level elements. When a user expands an element with a mouse click, only the next lower level elements load. This ‘pessimistic’ loading extracts a small toll on each click, but does not make the user wait for the whole list to load and then for the browser to parse and display it.

After the final working prototype or first releasable version of the software is complete, a business process reengineering study should analyze the impact of the software, if any, on analyst’s productivity.

This thesis discussed simulation model file parsing in the abstract, then only implemented support for Extensible Markup Language (XML) models. Future research should include implementing database file support into the system.

Similarly, this thesis described implementation on a high-performance computing cluster (HPC) in the abstract, but the prototype works only with the Condor cluster controller. Altering the code to work with a different cluster controller should be trivial based on the architecture, but has potential to be highly profitable for other organizations including the Modeling, Virtual Environments, and Simulation (MOVES) Institute at the Naval Postgraduate School. Taking this a step further, research could focus on allowing access to HPC's at any cooperative organization, such as any one of the former Project Albert partners, that would adopt this architecture. One intriguing possibility is that the Condor controller can distribute workloads over a very broad and loosely affiliated network. Cooperating organizations could use this to greatly enhance their computing power by having a cross-facility sharing arrangement.

Participants in a SEED Center seminar strongly advocated adding an expert system to help the researcher select the most appropriate design. The expert system would take into account the factors that the researcher selected, the factor types, the level of response surface complexity the researcher requires and available time, and then provide the researcher with a recommended design.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Acts as Attachment [Computer Software]. Rick Olson Retrieved June 12, 2007 from http://weblog.techno-weenie.net/articles/acts_as_attachment

Aptana RadRails Version 0.7.2 [Computer Software]. Open Source, Retrieved May 13th 2007 from http://www.aptana.com/

Aqtree Version 2 [Computer Software]. Open Source, Retrieved June 7th 2007 from http://www.kryogenix.org/code/browser/aqtree2/

Atkinson, L. (2001, October 3). Storing files in a relational database. Retrieved June 5, 2007, from http://www.zend.com/zend/trick/tricks-sept-2001.php

Bass, Len, Paul Clements, and Rick Kazman. Software Architecture in Practice. 2nd ed. Boston: Addison-Wesley, 2003.

Chwif, L.; Barretto, M.R.P.; Paul, R.J., (2000). On Simulation Model Complexity [Electronic version]. Winter Simulation Conference Proceeding, 1(1), 255-449.

Cioppa, T. M. (2002). Efficient Nearly Orthogonal and Space-Filling Experimental Designs for High-Dimensional Complex Models. Ft. Belvoir: Defense Technical Information Center. http://handle.dtic.mil/100.2/ADA406957

Crystal Ball for Microsoft Excel Version 2000 [Computer Software], Thousand Oaks, CA: EMAGENIT.

Defense Modeling and Simulation Coordination Office. (2007). Modeling and Simulation Analysis Center Journal, 2-3. Retrieved September 12, 2007, from http://www.dod-msiac.org/pdfs/newsletter/msiacnewsletter_06_2007.pdf

Dempsey, James. MVC Song. World Wide Developers Conference, 2003. Retrieved September 12th 2007 from http://cakephp.org/ModelViewController.mp3

Department of Defense. Under Secretary of Defense for Acquisition and Technology (1995) Directive 5000.59-P, Modeling and Simulation (M&S) Master Plan.

Eclipse Version 3.2.0 [Computer Software]. Portland OR: Eclipse Foundation, Inc.

JMP Version 7 [Computer Software]. Cary, NC: SAS Institute Inc.

Kennard, R. W. & Stone, L.A. (1969). Computer Aided Desing of Experiments. Tecnometrics, 11(1), 137-148.

Kleijnen, J. P. (2003). A user's guide to the brave new world of designing simulation experiments. Discussion paper, no. 2003-01. Tilburg: Tilburg University.

Law, A. M., & Kelton, W. D. (1982). Simulation modeling and analysis. McGraw-Hill series in industrial engineering and management science. New York: McGraw-Hill.

Libxml Version 0.3.8.4 [Computer Software]. Open Source, Retrieved May 15th, 2007 from http://rubyforge.org/projects/libxml

Lighttpd Version 1.4.15 [Computer Software]. Open Source, Retrieved May 15th 2007 from http://www.lighttpd.net/

Live Tree [Computer Software]. Open Source, Retrieved August 14th, 2007 from http://wiki.rubyonrails.org/rails/pages/LiveTree

Mongrel Version 1.0.1 [Computer Software]. Open Source, Retrieved May 15th 2007 from http://mongrel.rubyforge.org/

Mossberger, D., & Jin, T. (1998). Httperf: A tool for Measuring Web Server Performance. Performance Evaluation Review, 26(3), 31-37.

Netbeans Version 5.5.1 [Computer Software]. Santa Clara, CA: Sun Microsystems.

Process Modeler Version 7.0 [Computer Software]. Santa Clara, Ca: Savion

Project Albert (2005). Retrieved June 19, 2007, from http://www.projectalbert.org/index.html

Rauhut, M. W. (1999) Automating a Study Question Methodology to Enhance Analysis in High Level Architecture. Unpublished master’s thesis, Naval Postgraduate School, Monterey, CA.

Rockwell Arena Version 11.0 [Computer Software]. Milwaukee, WI :Rockwell Automation.

Ruby Development Tools Version 0.8 [Computer Software]. RubyPeople Retrieved May 15th 2007 from http://rubyeclipse.sourceforge.net V0.8

Rusco, M. S. (2003, August 8). Using Simulation Software for Design of Experiment Training. Retrieved July 19, 2007, from http://www.statease.com/pubs/dragracing.pdf

Sanchez, S. NOLH designs spreadsheet. Version 4 [Computer Software] http://diana.cs.nps.navy.mil/SeedLab/ [accessed 03/05/2007]

Subclipse Version 1.2.0 [Computer Software]. Open Source, Retrieved May 14th 2007 from http://subclipse.tigris.org/

The Free On-line Dictionary of Computing. Retrieved September 08, 2007, from http://dictionary.reference.com/browse/usability

Tiller [Computer Software]. Honolulu, HI: Referentia.

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

Clements, P., Kazman, R., & Klein, M. (2002). Evaluating software architectures: methods and case studies. SEI series in software engineering. Boston: Addison-Wesley.

Fowler, M., & Scott, K. (2000). UML distilled: a brief guide to the standard object modeling language. Reading, Mass: Addison Wesley.

Nance, R. E., & Sargent, R. G. (2002). ARTICLES - Perspectives on the Evolution of Simulation. Operations Research. 50(1), 161.

Thomas, D., Fowler, C., & Hunt, A. (2004). Programming Ruby: The Pragmatic Programmers' Guides (2nd ed.). Raliegh: Pragmatic Bookshelf.

Thomas, D., & Heinemeier-Hansson, D. (2006). Agile Web Development with Rails (2nd ed.). Raliegh: Pragmatic Bookshelf.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Paul Sanchez
   Naval Postgraduate School
   Monterey, California

4. Mr. Jack Jackson
   TRAC Monterey
   Monterey, California

5. MAJ John Alt
   TRAC Monterey
   Monterey, California